DÜZCE ÜNİVERSİTESİ | LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

DAĞITIK HADOOP KÜMELERİNDE YENİ EŞLE/İNDİRGE PROGRAMLAMA ALGORİTMASI MODELİ

EMİN ŞEŞEN

DOKTORA TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMAN
PROF. DR. RESUL KARA

DÜZCE, 2024

T.C.
DÜZCE ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

DAĞITIK HADOOP KÜMELERİNDE YENİ EŞLE/İNDİRGE PROGRAMLAMA ALGORİTMASI MODELİ

Emin ŞEŞEN tarafından hazırlanan tez çalışması aşağıdaki jüri tarafından Düzce Üniversitesi Lisansüstü Eğitim Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **DOKTORA TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Prof. Dr. Resul KARA

Düzce Üniversitesi

Jüri Üyeleri

Prof. Dr. Resul KARA
Düzce Üniversitesi _____

Dr. Öğr. Üyesi Serdar KIRIŞOĞLU
Düzce Üniversitesi _____

Dr. Öğr. Üyesi Nihan KAZAK ÇERÇEVİK
Bilecik Şeyh Edebali Üniversitesi _____

Doç. Dr. Süleyman UZUN
Sakarya Uygulamalı Bilimler Üniversitesi _____

Dr. Öğr. Üyesi Enver KÜÇÜKKÜLAHLI
Düzce Üniversitesi _____

Tez Savunma Tarihi: 27/06/2024

BEYAN

Bu tez çalışmasının kendi çalışmam olduğunu, tezin planlanmasından yazımına kadar bütün aşamalarda etik dışı davranışımın olmadığını, bu tezdeki bütün bilgileri akademik ve etik kurallar içinde elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara kaynak gösterdiğimi ve bu kaynakları da kaynaklar listesine aldığımı, yine bu tezin çalışılması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını beyan ederim.

27 Haziran 2024

Emin ŞEŞEN

TEŞEKKÜR

Doktora öğrenimimde ve bu tezin hazırlanmasında gösterdiği her türlü destek ve yardımdan dolayı çok değerli hocam Prof. Dr. Resul KARA'ya en içten dileklerimle teşekkür ederim.

Tez çalışması süresince, tez izleme komitelerinde değerlendirmeleri ile desteklerini esirgemeyen Dr. Öğr. Üyesi Serdar KIRIŞOĞLU ve Dr. Öğr. Üyesi Nihan KAZAK ÇERÇEVİK hocalarıma en içten dileklerimle teşekkür ederim.

Bu çalışma sırasında sabırla, desteklerini esirgemeyen sevgili eşim Gülderen'e, kızlarım Gökçe ve Ayça'ya en kalbi duygularımla teşekkürlerimi sunarım.

27 Haziran 2024 **Emin ŞEŞEN**

İÇİNDEKİLER

Sayfa No

ŞEKİL LİSTESİ

Sayfa No

ÇİZELGE LİSTESİ

Sayfa No

KISALTMALAR

AM	Uygulama Yöneticisi
DC	Veri Merkezi
DCG	Veri Merkezi Geçidi
DCM	Veri Merkezi Denetleyici
DFS	Dağıtık Dosya Sistemi
EKK	En Küçük Kareler Yöntemi
GRG	Küresel İndirgeme Grafı
HDFS	Hadoop Dağıtık Dosya Sistemi
HTTP	Hiper Metin Transfer Protokolü
IoT	Nesnelerin İnterneti
IP	İnternet Protokolü
JVM	Java Sanal Makinesi
NM	Düğüm Yöneticisi
RAM	Rastgele Erişimli Bellek
RM	Kaynak Yöneticisi
VM	Sanal Makine
YARN	Başka Kaynak Müzakerecisi

ÖZET

DAĞITIK HADOOP KÜMELERİNDE YENİ EŞLE/İNDİRGE PROGRAMLAMA ALGORİTMASI MODELİ

Emin ŞEŞEN

Düzce Üniversitesi
Lisansüstü Eğitim Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı
Doktora Tezi
Danışman: Prof. Dr. Resul KARA

Haziran 2024, 67 sayfa

Büyük veriler veri aktarma maliyetinden dolayı genellikle üretildiği konumlara yakın yerlere depolanırlar. Depolanan bu veriler işlenmek için tek bir konuma taşınır veya bulunduğu konumda işlenirler. Literatürde veri işlemek için farklı yöntemlere rastlamak mümkündür. Bu çalışmada veri işlemek için yeni bir yöntem sunulmuştur. Önerilen yöntemde veri işleme sürecini tamamlayan farklı donanımlara sahip veri merkezlerinin (DC) kendi aralarında veri karıştırma (shuffling) yapması sağlanmıştır. DC'lerin indirge (reduce) fonksiyonunun veri işleme maliyetinin hesaplanması için test ortamında elde edilen veriler ile polinomal regresyon modeli oluşturulmuş ve karar sürecinde bu modelden elde edilen katsayılar kullanılmıştır. Karıştırma yapılacak anahtar/değer çiftlerini, konumlarını dikkate alarak, DC'lerin maliyetlerine göre dağıtılmıştır. DC'ler arasında karıştırma için, DC'lerin tümünün işini bitirmesi beklenmez. Böylelikle tüm DC'lerin aynı anda karıştırma yaptıklarındaki hem karıştırma hem de işlenen veri hacmi azalmıştır. Önerilen yöntemin performansı literatürdeki 4 farklı yöntemle karşılaştırılmıştır. Sonuç olarak bu çalışma veri boyutunda en yakın örneğinden %15 daha az karıştırma verisi oluşturmuştur.

Anahtar Sözcükler: Büyük Veri, Dağıtık MapReduce, Dağıtık Veri Merkezleri, Veri Azaltma

ABSTRACT

NEW MAP/REDUCE PROGRAMMING ALGORITHM MODEL IN DISTRIBUTED HADOOP CLUSTERS

Emin ŞEŞEN
Düzce University
Graduate School, Department of Computer Engineering
Doctoral Thesis
Supervisor: Prof. Dr. Resul KARA

December 2022, 67 pages

Big data are often stored close to the locations where they are generated, owing to the cost of data transfer. These stored data are moved to a single location for processing or processed at that location. In the literature, it is possible to find different methods for processing data in distributed datacenters. In this study, we present a new method for data processing called GSelf-MapReduce. In the proposed method, shuffling is performed among heterogeneous datacenter (DC) that complete the data-processing process. To calculate the data processing cost of the reduced function of the DCs, a polynomial regression model was created using the data obtained in the test environment, and the coefficients obtained from this model were used in the decision process. The key/value pairs to be shuffled are distributed according to the cost of the DCs, considering their location. Because the data to be shuffled between DCs do not wait for all DCs to complete their jobs, the cost is reduced both in terms of the data to be moved and the data to be processed. The performance of the proposed method was compared with that of four different distributed data processing methods in the literature. As a result, this work generates 15% less shuffled data than the closest work.

Keywords: Big Data, Data Reduction, Geo-distributed MapReduce, Heterogenous Datacenters

EXTENDED ABSTRACT

NEW MAP-REDUCE PROGRAMMING ALGORITHM MODEL IN DISTRIBUTED HADOOP CLUSTERS

Emin ŞEŞEN
Düzce University
Graduate School, Department of Computer Engineering
Doctoral Thesis
Supervisor: Prof. Dr. Resul KARA

June 2024, 67 pages

1. INTRODUCTION

Big data has emerged as a new research field in academia and industry. High-volume, rapidly increasing data per unit time are called big data. These include social network interactions, electronic mail data, videos, sensor data, images, search engine queries, health records, data generated from customers, and access records produced by large companies. Because the generated data do not have a specific structure, they are not suitable for storage in a structured database.

Hadoop is a scalable and reliable software project that enables large datasets to be stored and computed on distributed nodes. It can consist of one or more servers, each of which performs local data storage and computational computations. Hadoop is preferred for large data storage because of its scalability, high availability, fault tolerance, low cost, and efficiency.

Let us imagine that a heavily used web application server can serve different geographical locations. This server generates access information (IP, HTTP status code, browser, file path), access errors, and data-load log data. Because the size of this log data is large, it can be stored in a Hadoop cluster near an application server. It is very difficult to move and process these data in a single data center. This results in the intensive use of network resources around aggregated datasets and increases transmission time. Therefore, instead of centralized data aggregation, an ideal method is to aggregate geographically distributed data from multiple sources in multiple-edge DCs and process them in parallel in the DC where they are stored. However, Hadoop does not provide any infrastructure to perform MapReduce processing of unstructured data stored on multiple Hadoop clusters.

Organizations with multiple DCs may have multiple data processing nodes. Nodes in one

datacenter can be very different from nodes in another datacenter in terms of CPU frequency, number of cores, cache size, memory size, network bandwidth, and storage capacity. These factors are important for the performance of computations on the datasets. If the dataset to be processed is small and the time to transfer the data is insignificant, the datasets can be transferred to a single data center for processing.

We propose a method for processing data in distributed datacenters using MapReduce. In the stages of the proposed model, the job progress of DCs that have completed their processing and the datasets that are still being processed are checked. The status of DCs is determined by job progress. For the distribution of keys between clusters, the reduction function data that the dataset processes per unit time is included in the parameters. To calculate the distribution of keys, the keys were grouped according to the DCs. Thus, the distribution was provided according to the number of keys.

2. MATERIAL AND METHODS

Hadoop does not provide an interface for the MapReduce programming model in its distributed clusters. The MapReduce function can be executed in distributed clusters using different approaches. Geo-Hadoop is an approach in which data are stored in different datacenters and processed distributedly. Geo-Hadoop is categorized under two architectures: centralized and decentralized. A centralized architecture, a manager node located in one of the clusters, manages the resources of the other cluster nodes. In a decentralized architecture, each cluster manages its own worker node. Each cluster can run Hadoop jobs independently. In addition, clusters can share computation and data resources with each other for geographically distributed jobs. In hierarchical Hadoop, each cluster completes a MapReduce task. The key/value outputs of the clusters are transferred to a single Global Reducer. The output was obtained by running the reduce function in Global Reducer. Although the MapReduce-Global Reducer approach has a two-layer structure, it can be adapted to hierarchical approaches with more layers, as needed.

In a study called Cross-MapReduce, these two methods were combined, and a method called GShuffling was presented. As shown in Figure 2, after the map function is completed in multiple nodes in a cluster in geo-Hadoop, multiple key/value pairs with the same key are moved between clusters. In this study, after the MapReduce job is completed in each cluster, keys are exchanged between clusters and only between datacenters where

the key is available. Thus, it aims to prevent unnecessary network traffic in data centers that do not have a key/value pair.

In the proposed GSelf-MapReduce method, when the clusters complete data processing, they are allowed to shuffle among themselves. Therefore, it is not expected that all clusters will complete their jobs. For example, Let's consider a structure of 6 clusters. Because the data in these clusters may contain data of different sizes and the computational capacities of the clusters may be different, the running times of the MapReduce function may also differ.

If the MapReduce job in two Clusters is completed before the others, they exchange data by shuffling among themselves. After shuffling, the reduce function is run on clusters this clusters. Subsequently, after The MapReduce job in at least two clusters completed, that clusters shuffling and then all clusters, the reduce function is run in the same way. To shuffle between clusters, the method in Cross-MapReduce was modified. The distribution of the keys/values occurs according to the computational capacity of the clusters. For the computational capacity, job type, reduce the input data size, reduce input key count, reduce output data size, reduce output key count, reduce output data size, reduce output key count, RAM, and CPU values of the node where the reduce function is running and the running time of the function are considered. A sixth-order polynomial regression model was used to calculate the coefficients showing the relationship between this variables. The system was trained using 75% of the test data, and the prediction performance of the system was tested using 25% of the data.

Using the coefficients of the equation obtained by polynomial regression, the costs of the nodes in the clusters where the reduce function is applied are calculated.

For shuffling, keys are moved from the clusters that have completed their jobs to the cluster with the largest key size. Key distribution occurs according to polynomal equation. Subsequently, the shuffling process was initiated. After shuffling was completed, the reduce function was executed on the clusters. Thus, the data volume and shuffling time in shuffling during data exchange between clusters and, therefore, job time are reduced.

3. RESULTS AND DISCUSSIONS

The proposed GSelf-MapReduce method is implemented in a distributed cluster environment. An environment consisting of four clusters was created for this purpose. In this environment, each cluster has a different hardware and data size. Clusters were

created on servers at Düzce University, each on different networks. Each server node was considered an independent datacenter. Each datacenter had a cluster. In addition to the clusters, DCM ensures the continuity of the workflows of the clusters.

The keys are then sent to multiple global reducers. In GSelf-MapReduce, keys are sent from a cluster with a small key volume to a cluster with a large total key volume. The job progress of the clusters that continued their jobs was checked. If the progress is less than the predefined threshold value, the key distribution of the clusters that have completed their job is determined according to the running performance of the reduce function. Keys are sent to the clusters. According to these distributions, the clusters perform shuffling. After the shuffling process is completed, the reduce function is executed on the data in these clusters (Phase-1). Then, the processes in phase-1 are performed until all clusters complete their jobs.

In the Cross-MapReduce, Hierarchical and Geo-Hadoop approaches, shuffling is performed when all clusters are completed. Because GSelf-MapReduce can have multiple phases, we compared the data volume realized in the last phase of shuffling with the data volume of the other approaches.

The data processing time in the word count application is approximately the same as that in our proposed GSelf-MapReduce and Hierarchical approach. The most important reason is that the key volume is close to the key/value volume. This is because grouping and distributing keys in a cluster and moving these keys to clusters incurs time costs. Although the cross-MapReduce approach produces less shuffling volume than the hierarchical approach, its production time is higher. In addition, the shuffling volume of the proposed method is 79% less than that of Cross-MapReduce, which is the closest method in terms of volume. This is because 66% of the total data were processed in the first phase.

The data-processing time of GSelf-MapReduce in the inverted index was shorter than that of Cross-MapReduce. In addition, the shuffling data volume was 69% smaller than that of the same method. Cross-MapReduce is followed by the Hierarchical and Geo-Hadoop approaches. The Geo-Hadoop shuffling data volume was significantly larger than the input data volume. This creates a burden on the network traffic, which considerably increases the data-processing time. Moreover, in the first phase of GSelf-MapReduce, 68.9% of the total data were reduced by processing.

The closest approach to shuffling the data volume in the adjacency list of GSelf-MapReduce is Cross-MapReduce. It can be seen that the final phase results in about 15% less data volume than Cross-MapReduce. The reduced adjacency list phase requires intensive data processing. Because the volume of the shuffled data directly affects the volume of the data that the reducers process, the processing time of the data increases. An example is the hierarchical approach. This is because the data moved from the clusters are moved to a single cluster and processed by the global reducer. In Geo-Hadoop, on the other hand, because of the participation in the output shuffling of the Map phase, the volume of data coming out of the clusters is very large. In addition, the proposed method processed 44.6% of the total data in the first phase, reducing the data volume to be processed.

One of the advantages of GSelf-MapReduce over Cross-MapReduce in terms of time is that no time is spent creating GRGs. Instead, it is sufficient to determine the cost of machines. The keys of the clusters in the common key groups were distributed proportionally to these costs. In addition, the data-processing costs of the clusters were considered.

Clusters with different hardware specifications and data volumes have different data processing times. While Cross-MapReduce, Hierarchical Hadoop, and Geo-Hadoop, which are distributed MapReduce data processing approaches, wait for all clusters to complete their job, in the proposed approach, clusters with progress greater than a certain threshold are shuffled among themselves and then reduced. Thus, the amount of data to be processed was reduced. Thus, clusters that are slower, have larger data volumes, and have lower processing speeds are not expected to complete their jobs. This reduces the shuffling traffic and processing time for all clusters.

4. CONCLUSION AND OUTLOOK

Storing and processing data in distributed datacenters is a necessity for this age. This is because it is important to reduce the cost of data transportation by storing data at the location where it is generated. It may be desirable to obtain a result by processing the distributed data. MapReduce is one of the methods proposed for processing distributed data. In this study, a method called GSelf-MapReduce was proposed based on Cross-MapReduce. A similar but more efficient approach is presented for distributing the keys in cross-MapReduce by considering the heterogeneity of clusters. Using this method,

clusters are shuffled among themselves by examining their job progress. Thus, not all clusters are expected to complete the data processing. The data is shuffled. After shuffling is completed, a reduction operation occurs. Because this operation is performed, the amount of data to be shuffled in the next stage is reduced. It also reduces the data to be processed for jobs that need to run an intensive reduction function. Therefore, the runtime of the reduced function is reduced.

The proposed GSelf-MapReduce method is compared with Geo-Hadoop, Hierarchical and Cross-MapReduce. In this study, word count and inverted index adjacency-list applications were used for comparison with other methods, and their advantages in terms of time and volume of data to be shuffled.

1. GİRİŞ

Günümüzde akademik ve endüstriyel alan için yeni bir araştırma çalışma alanı olarak büyük veri ortaya çıkmıştır. Birim zamanda yüksek hacimli, hızla artan veriler, büyük veri olarak adlandırılır. Bunlar, sosyal ağ etkileşimleri, elektronik posta verileri, videolar, algılayıcı verileri, resimler, arama motoru sorgulamaları, sağlık kayıtları, büyük şirketlerin ürettiği müşteri ve erişim kayıtlarından üretilen verilerdir [1], [2], [3]. Veriler belirli bir düzene sahip olmadığından, yapısal veri tabanında depolamaya elverişli değildir. Bu verilere bir örnek vermek gerekirse, 2022 yılında günlük ortalama olarak; Youtube'a 720 bin saatlik içerik yüklenmekte [4], Google'da 8,5 milyar arama yapılmakta [5], Instagram'da 3 milyar fotoğraf yüklenmekte [6], Facebook'da ise 5,75 milyar beğeni ve 4,3 milyar mesaj gönderilmektedir [7]. 2025'e kadar üretilen verinin 180 ZB olarak olacağı tahmin edilmektedir [4]. Bu verilerin yapısal veri tabanında depolanıp, analiz edilmesi mümkün değildir.

Hadoop büyük veri kümelerinin dağıtık sunucular üzerinde depolanmasını ve hesaplanmasını sağlayan ölçeklenebilir bir yazılım projesidir. Her biri yerel veri depolama ve hesaplama yapan bir veya daha fazla sunucudan oluşabilir [8]. Hadoop ölçeklenebilirlik, yüksek kullanılabilirlik, hata toleransı, düşük maliyet ve verimlilik gibi özelliklerinden dolayı büyük veri depolama için tercih edilmektedir.

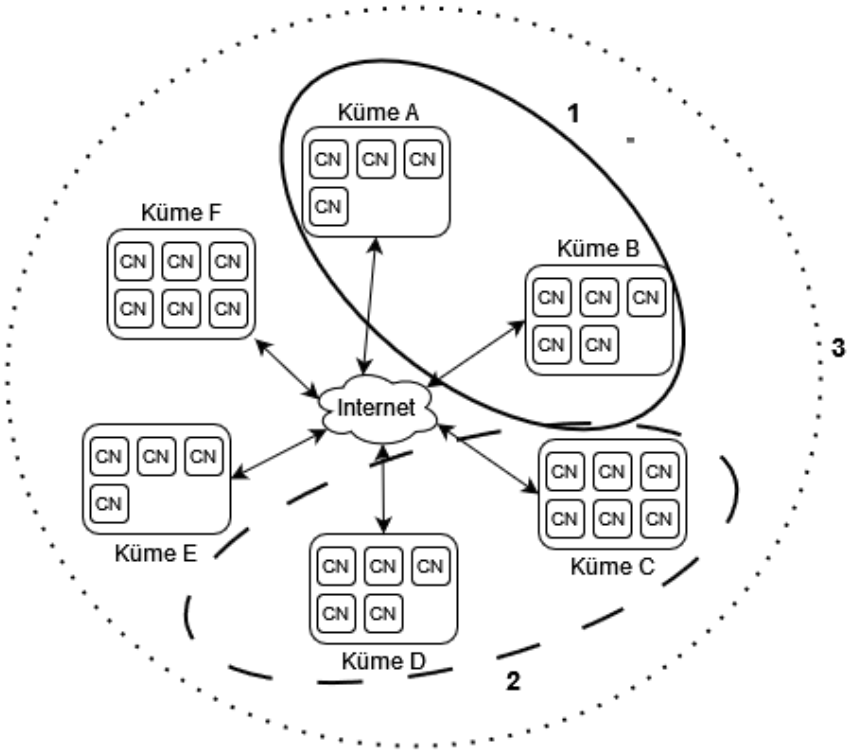
Yoğun kullanılan bir web uygulama sunucusunun, farklı coğrafi konumlarda hizmet verdiğini düşünelim. Bu sunucu; erişim bilgileri (IP, HTTP durum kodu, tarayıcı, dosya yolu), erişim hataları, veri yükleme gibi günlük verileri üretir. Günlük verilerinin boyutu büyük olduğundan uygulama sunucusunun yakınında depolanabilir. Verilerin tek bir veri merkezine (DC) taşınıp işlenmesi oldukça zordur [9], [10]. Çünkü bu durum, kümeler etrafındaki ağ kaynaklarının yoğun kullanımına neden olur ve verilerin iletim süresini arttırır. Bu nedenle, merkezi veri toplama yerine, birden çok coğrafi olarak dağılmış kaynaklardan gelen verileri, birden fazla uç DC'de toplamak ve bunların depolandığı DC'de paralel bir şekilde işlemek ideal bir yöntemdir [11]. Fakat Hadoop; birden fazla Hadoop kümesi üzerinde depolanan yapısal olmayan verilerin MapReduce (eşle/indirge) işleme süreçlerini gerçekleştirmek için herhangi bir altyapı sağlamamaktadır.

Birden fazla DC'ye sahip kuruluşlar birden fazla veri işleme düğümüne sahip olabilir. Bir DC'de düğümler, CPU frekansı, çekirdek sayısı, önbellek boyutu, bellek boyutu, ağ bant genişliği ve depolama kapasitesi açısından başka DC'deki düğümlerden çok farklı olabilir. Bu faktörler kümelerdeki hesaplamaların performansları için önemlidir. Eğer DC'lerde işlenecek veri seti küçükse ve veriyi aktarmak için harcanacak zaman önemsiz olduğunda, veri setleri tek bir veri merkezine aktarılarak işlenebilir [12].

Büyük verinin, veri kayıplarına karşı yedekliliğinin sağlanması da önemlidir. Ayrıca boyutu sebebiyle de tek bir düğümde saklanması mümkün değildir. Bu gibi nedenlerle dağıtık depolama ve analiz etme ihtiyacı ortaya çıkmıştır.

1.1. ÇALIŞMANIN AMACI

Hadoop, dağıtık kümelerde eşle/indirge programlama modeli için herhangi bir altyapı sunmaz. Dağıtık kümelerde farklı yaklaşımlarla, eşle/indirge fonksiyonu çalıştırılabilir. Şekil 1.1'de önerilen yaklaşım verilmiştir.



Şekil 1.1. Önerilen yaklaşım.

Önerilen yöntemde, kümeler veri işleme sürecini bitirdiklerinde, kendi aralarında karıştırma yapmaları sağlanır. Böylelikle kümelerin hepsinin işini bitirmesi beklenmez.

Örneğin, Şekil 1.1'de altı tane küme bulunmaktadır. Bu kümelerdeki düğümler üzerinde farklı boyutta veriler yer alabilir ve kümelerin hesaplama kapasiteleri farklı olabileceğinden, eşle/indirge fonksiyonunun çalışma süreleri de birbirinden farklı olabilir.

Şekil 1.1'de 1. durumda A ve B kümelerindeki iş diğerlerinden daha önce bittiğinden kendi aralarında karıştırma yaparak veri değişimini gerçekleştirirler. Karıştırmadan sonra A ve B kümelerinde indirge işi çalıştırılır. Böylelikle bu iki küme arasındaki ortak anahtar/değerler çiftleri, diğer kümelerin işlerini bitirmesini beklemeden azaltılır. Sonrasında ise 2. durumdaki kümeler (C ve D) ve 3. durumdaki tüm kümeler karıştırma gerçekleştirdikten sonra, aynı şekilde indirge fonksiyonu çalıştırılır. Kümeler arasındaki karıştırmada anahtarların konumları dikkate alınır. Karıştırmada taşınacak anahtar/değerler dağılımları, kümelerin hesaplama kapasitesine göre değerlendirerek gerçekleştirilir.

Önerilen yöntemde, kümeler kendi aralarında karıştırma yaptıklarından son aşamadaki tüm kümelerin aralarında oluşan yoğun karıştırma trafiği azaltılmış olur. Böylelikle karıştırma veri hacmi ve işin toplam tamamlanma süresi kısalır. Ayrıca kümeler kendi aralarındaki anahtarlar tekilleştirildiğinden sonraki aşamalarda, bu kümeler arasında karıştırma gerçekleşmez.

1.2. LİTERATÜRDEKİ ÇALIŞMALAR

Hadoop, eşle/indirge ile veri işlemede, bir veya daha fazla düğümden oluşan kümeler üzerinde iyi bir performans göstermesine rağmen coğrafi olarak dağıtık yapıdaki kümelerde iyi bir performans göstermez. Literatürde, dağıtık kümeler üzerinde veri işlemenin iyileştirilmesi için yapılan çok sayıda çalışmalar bulunmaktadır.

Quoc ve arkadaşları, internet üzerindeki web sitelerini tek bir makine ile taramak zor olduğundan coğrafi olarak dağıtık yapıda bot geliştirmişlerdir. Geliştirdikleri sistem ile her bir küme, başlangıçta verilen bağlantılardan başlayarak internet tararlar. Yeni bulunan bağlantıları ve verileri ayrıştırıp sisteme eklerler ve periyodik olarak bu işlemi gerçekleştirerek verileri güncel tutarlar. Bu sistemde geliştirdikleri alma ve parçalama yöntemleriyle, kümeler arası veri iletişim maliyetini düşürmeyi amaçlamışlardır [14].

Wang ve arkadaşları, coğrafi olarak dağıtılmış farklı donanıma sahip kümelerde, eşle/indirge işleri için kullanılan enerji maliyetinin düşürülmesini amaçlamışlardır.

Verilerin işlenmesi için verinin konumu, işin bitme süresi ve kaynak yeterliliği ele alınarak bir iş zamanlama çerçevesi (DTS) önermişlerdir [15].

Imai ve arkadaşları, IoT verilerinin işlenebilmesi için katmanlı bir mimari sunmuşlardır. Bu katmanlı mimarilerin seviyelerine göre verinin bulunduğu en alt seviye düğümden üste doğru eşle/indirge sonuçlarının indirge fonksiyonuyla birleştirilmesi önerilmiştir. Hiyerarşik veri toplamayla sorgu süresinin azaltılması amaçlanmıştır [16].

Xiao ve arkadaşları, dağıtık kümelerde çalışan eşle/indirge uygulamalarının; eşle ve indirge aşamaları arasındaki bant genişliği, bilgi işlem, verilerin taşınması ve depolama maliyetlerinin birlikte düşürülmesi amaçlanmıştır. Lyapunov optimizasyon algoritması ile, veri hareketi, veri kaynağı ve indirge düğümlerinin seçimi için bir yöntem önerilmiştir. [17].

Nintyananham ve arkadaşları, dağıtık veri merkezlerinde, veri işlemede kaynak ve maliyet optimizasyonu yapılarak eşle/indirge işlerinin verimliliğini artırmayı amaçlamışlardır. Ateş böceği sürü optimasyon algoritması ile bant genişliği depolama kapasitesi, enerji ve hesaplama maliyetleri amaç fonksiyonları olarak belirlenmiştir. Eşle/indirge işleri için uygun veri merkezleri belirlenerek iş yüklerinin azaltılması sağlanmıştır [18].

Gousasmi ve arkadaşları, dağıtık veri işlemede, bulut sağlayıcıların karını artırmayı amaçlayan bir iş planlama algoritması tasarlamışlardır. Veri yerelliğinden yararlanarak bulut sağlayıcıların boşta bulunan kaynakların, kullanıcıların gönderdikleri işlerin tamamlanma süresi ile birlikte hem VM maliyeti hem de aktarım maliyetlerini göz önünde bulundurularak, verilerin yerelde veya uzak kümelerde çalıştırılmasını için; karar veren bir yöntem önermişlerdir [19].

Qin ve arkadaşları, dağıtık veri merkezlerindeki hem veri merkezi içi ağ ve sunucu kaynaklarının kullanımından oluşan, hem de veri merkezleri arası veri taşıma nedeniyle oluşan enerji maliyetlerini azaltılması amaçlanmıştır. Farklı donanımlara sahip veri merkezlerindeki düğümlerin depolama kapasitesi bellek kapasitesi, işlem gücü değişkenleri ve iş yükü çeşitliliği ile Lyapunov optimizasyon yöntemi kullanılarak enerji tüketimine bağlı karbon emisyonunu azaltmışlardır [20].

Wang ve arkadaşları, dağıtık veri merkezindeki kaynak sağlayıcıların karlarının artırmayı amaçlayan, veri yerelliğine ve işin tamamlanma süresine dayanan bir eşle/indirge iş akış yöntemi önermişlerdir. Eşle/indirge işlerin en kısa sürede tamamlanması için otomatik

programlama yapan bir iş akışı geliştirilmiştir. Görevlerin, girdilerin ve çıktılarının iletim sürelerini azaltacak şekilde planlanması sağlamışlardır [21].

Yibo ve arkadaşları, verilerin; kümeler arasındaki iletim süresini ve kümelerde çalışma süresini en aza indirmeyi amaçlamışlardır. Kümelerde çalışacak eşle/indirge işleri; eşle ve indirge görevleri olarak, kümelerdeki iş yüklerine ve kümeler arası ağ yoğunluğuna göre ayrı ayrı ele alarak zamanlama algoritması tasarlamışlardır. Kümelerdeki işler kendi içinde kuyruklarda bekletilir. Kuyruklardaki işlerin tahmini bekleme süresi kümeler için tasarlanan küresel kontrolcüye bildirilir. Kontrolcü verilerin işleneceği kümelere karar verir [22].

Liu ve arkadaşları, dağıtık kümelerde, işlerin tamamlanma süresini azaltmak ve bant genişliğinde verimliliği sağlamak için çok aşamalı bir veri toplama ve işleme yöntemi sunmuşlardır. Dağıtık kümelerde veriler kendi üzerinde işlendikten sonra, ilk aşamada aralarında seçilen merkezi düğümlere taşınır. Birçok kenar düğüm için birden fazla küme seçilir. Burada işlenen veriler, son aşamada yine bir kümeye taşınarak işlenir [11].

Wang ve arkadaşları, DC'ler arası veri iletim süresini azaltmak için bir veri yerleştirme algoritması önerilmiştir. Mobil cihazlar üzerindeki veri işleme yükünün buluttaki Hadoop sunucularına taşınarak istemcilerdeki yükü azaltmışlardır. Ayrıca resim dosyaları dağıtık veri işleme yöntemleriyle daha kısa sürede analiz edilerek, istemcilere daha hızlı yanıt verilmesi amaçlanmıştır [23].

Atrey ve arkadaşları, yoğun hesaplama gerektiren verilerin, bulut ortamlarındaki DC'lere yerleştirilmesi için bir çerçeve önerilmiştir. Hadoop'da kullanılan hash tabanlı veri bölümleme yöntemlerine yerine iki algoritma önermişlerdir. Bu algoritmalardan biri tek bir makinede çalışabilen uygulamalar için verimlilik sağlamayı amaçlarken, diğer algoritma ise tek bir makineye sığmayan çok büyük iş yükleri için ölçeklendirme modeli sunar [24].

Li ve arkadaşları dağıtık veri merkezlerinde görevleri, çalışacağı makineler atandığında oluşan kaynak israfı veya kaynak yetersizliği problemlerini çözmek için işlerin planlanması ve işlenecek verilerin, veri merkezlerine yerleştirilmesi için yöntem önerilmiştir. Görev öncelikleri ve düğümlerin kaynakları hesaplanarak, görev atamaları gerçekleştirilir [25].

Wang ve arkadaşları, bulut bilişim sağlayıcılarının karının daha fazla olması için bir çalışma gerçekleştirmişlerdir. Veri yerelliğine dayanan, en kısa sürede eşle/indirge işini

bitirmeyi amaçlamışlardır. Eşle/indirge işlerinin iş türleri, işledikleri boyutları ve düğümlerin hesaplama kapasiteleri farklı olduğundan görevlerin atanacağı düğümler, verilerin aktarım süreleri, görevlerin işlenme süreleri ile doğrudan ilişkili eşle/indirge dinamik iş akış metodu sunmuşlardır [26].

Pandey ve arkadaşları, dağıtık veri merkezlerindeki enerji tüketimini en aza indiren bir iş planlama algoritması önermiştir. Hadoop YARN konteynerları üzerinde daha önceden çalıştırılmış eşle/indirge işlerinin enerji tüketimi, işlem süreleri ve hesaplama düğümlerinin özellikleri kaydedilerek profilleri oluşturulur. Oluşturulan profillere göre işin son teslim tarihini referans alarak yeni atanacak işler için en uygun veri merkezleri belirlenir [27].

Li ve arkadaşları, dağıtık veri merkezlerinde devam eden işler nedeniyle, farklı iş başlatma zamanlarını dikkate alarak zamanlama algoritması önermişlerdir. Her veri merkezinde, daha önceden çalıştırılan işlerin yürütülme süreleri elde edilerek, veri merkezlerinde ayrı iş kuyrukları oluşturulmuştur. Kuyruklama ve veri merkezlerindeki işlerin tamamlama zamanını belirlemek etmek için lagrange methodunu kullanmışlardır [28].

Lordache ve arkadaşları, bulut üzerinde eşle/indirge işlerinin yapılabilmesi için Amazon EMR API'si tasarlanmıştır. Birden çok ve farklı bulut kaynaklarının eş zamanlı kullanılabilmesi için amaçlanmıştır. Kullanıcı ile sistem arasında bir yazılım katmanı oluşturur. Farklı bulut kaynakları ile coğrafi olarak dağıtık hesaplama yapmaya imkân tanır [29].

Xiao çalışmasında, çok çekirdekli Hadoop kümelerinin eşle/indirge performansının artırılmasını amaçlamıştır. Bunun için aynı veri parçasının birden çok kez işlenmesini gerektiren yinelemeli eşle/indirge işlerinde, her yinelemeli işlem sırasında girdi verisinin düğümler arasında taşınması ile oluşan yoğun ağ ve disk trafiğinin önlenmesi için arka plan da çalışan bir önbellek tasarlamışlardır. Ayrıca çok çekirdekli Hadoop kümeleri üzerinde görevlerin paralel olarak çalıştırılmasında verimliliğin artırılması için, bir eşle/indirge işini; eşle ve indirge olarak ayrı işler olarak ayrıştırılıp, çalışması zamanı optimize edilmiştir [30].

Jayalath ve arkadaşları, dağıtık kümelerde birden fazla eşle/indirge işlerinin sıralı olarak, verimli şekilde gerçekleştirebilmek için bir çalışma yapmışlardır. Bu çalışmada DTG optimizasyon algoritması önermişlerdir. Bu algoritma parasal maliyet veya zamana göre

optimizasyon gerçekleştirmektedir. Dağıtık kümeler üzerindeki veri setlerinin bu optimizasyon algoritmasıyla analiz edilebilmesi için G-MR çerçevesini önermişlerdir. G-MR ile kümelerdeki dağıtık veriler üç farklı yol ile analiz edilebilir. Bunlar veri eşle aşamasından önce farklı kümeye taşınabilir, eşle aşamasından sonra; indirge aşaması farklı kümelerde ve eşle/indirge sonuçları farklı kümeye taşınarak gerçekleştirilebilir. Optimizasyon grafindaki kenarlar ya parasal maliyet ya da zamana göre belirlenir. Bu çalışmada seçilen optimizasyon hedefine göre, en kısa yol optimum yol olarak belirlenir [31].

Heintz çalışmasında, bağlantı hızları ve düğümlerin işlem gücüne göre, düğümlere görev ataması gerçekleştirir. Verinin eşle aşamasına giriş yapmasından başlayarak, eşle ve indirge işlemleri dahil 3 aşamada, eşle/indirge işleminin süresinin azaltılması amaçlanmaktadır. Bu aşamalar birbiriyle bağlantılı olarak değerlendirmişlerdir. Her fazın minimum sürelerinden ziyade 3 fazı bütünüyle değerlendirerek uçtan uca üretim süresini minimuma indirmişlerdir [32].

Luo ve arkadaşları, eşle/indirge işlerini koordine ederek çalıştırmak için, Hiyerarşik yaklaşım modeli benimsenerek dağıtık Hadoop kümeleri üzerinde zamanlama algoritmaları önermişlerdir. Çalışmayla iki zamanlama algoritması sunulur. Yoğun hesaplama kaynağı kullanan işler için, 'Hesaplama Kapasitesine Dayalı Zamanlama Algoritması'(CCAS) ve büyük veri setleri için ise 'Veri Konumuna Bağlı Hesaplama Algoritmasıdır' (DLAS) [12].

Xu ve arkadaşları, dağıtık kümelerdeki çok sayıda veri analizi işlerinin ağ maliyetini en aza indirmek için bir çalışma yapmışlardır. Çok sayıda veri işleme işinin performansını arttırmayı amaçlanmıştır. Hem veri işleme performansını ve ağ maliyeti problemlerini gidermek için Lyapunov optimizasyon teknikleri kullanmışlardır. Aynı zamanda gelecekteki sorgu isteklerine optimal sonuçlar sunması için, girdi verilerinin yerinin belirlenmesi amaçlanmıştır [33].

Chen ve arkadaşları, paralel veri işleyen veri merkezlerindeki veri analitiğinin performansını arttırmak ve veri merkezleri arasındaki veri akışını düzenlemek için bir çalışma yapmışlardır. Veri akışını doğrudan veri merkezlerine yönlendirmenin her zaman verimli olmayacağını alternatif veri merkezleri üzerinden atlamalı olarak gerçekleştirildiğinde verimli olabileceği değerlendirilmiştir [34].

Pu ve arkadaşları, coğrafi olarak dağıtılmış verilerin analiz edilerek istenilen sorguların

çalışma süresini en aza indirmeyi amaçlamışlardır. Kümeler arasındaki iki yönlü bant genişliği ile, giriş verilerinin konumu, eşle fonksiyonunun çıktısının konumu veya görevin çalışacağı kümelerin konumunu dikkate alarak en iyi sorgu süresini sağlayan bir yöntem sunmuşlardır. Bu süreyi değerlendirirken ağın parasal veri taşıma maliyetini göz önünde bulundurmuşlardır [10].

Luo ve arkadaşları, büyük veriler için, hiyerarşik yaklaşımda, kümelerdeki düğümlerin hesaplama kapasitesi ile bir algoritma önerilmiştir. Bunun için düğümlere girdi ve çıktı veri boyutlarının oranı ve bir düğümün birim zamanda işlediği girdi veri boyutu ile uygulama profili oluşturulur. Uygulama profillerine göre verinin düğümler arasındaki en kısa çalışma yolunu bulabilmek için düğümler arasında çok sayıda kombinasyon oluşturulur. İş, bu yolda çalıştırıldıktan sonra veriler, verimi (throughput) en yüksek küresel indirgeme sunucusuna yönlendirilerek iş tamamlanır. [12].

Chen ve arkadaşları coğrafi olarak dağıtılmış nesnelerin interneti (IoT) kümelerinin, veri analitiği maliyetini iyileştiren MCGL adında bir algoritma önerilmiştir. Kümelerdeki parasal maliyet, donanım kaynaklarının ve veri boyutu farklılıkları ele alınmış; işin yürütme süresinin optimizasyonu, parasal maliyetin optimizasyonu ve her ikisinin optimizasyonu birlikte değerlendirilmiştir. Görevlerin kümelere yerleştirilmesinde, kümelerin eşle, karıştırma ve indirge aşamalarındaki iş yükü, boş slot sayısı ile aralarındaki bant genişliği kullanılmıştır [35].

Cavallo ve arkadaşları, düğümlerin hesaplama kapasitesi ve bağlantıların kapasitesini göz önünde bulundurarak bir Hiyerarşik Dağıtık Hadoop Framework algoritması önermişlerdir. Kümeler üzerinde çalışan özgün Hadoop altyapısında herhangi bir değişiklik yapılmamıştır. Merkezi bir iş yöneticisi, düğümler üzerinde benzer işlerin küçük örnekleri çalıştırılır. İşlerin türüne göre düğümler için bir uygulama profili oluşturulur. Uygulama profili oluşturulan düğümlerde, gönderilecek işin tahmini süresi hesaplanır. Bu sürelerle birlikte verilerin farklı düğümlere taşınıp çalıştırılması değerlendirilir. Bu tahminlerle, çok sayıda kombinasyon oluşturulur. Kombinasyonlardan, işlenecek düğümler ile bağlantı yolları birlikte değerlendirilerek, işin en uygun çalışma yolu belirlenerek iş, kümeler üzerinde çalıştırılır. Sonuçlar küresel indirgeyici olarak belirlenen bir küme üzerinde toplanıp, indirge fonksiyonu çalıştırılarak nihai sonuçlar elde edilir [36].

Wang ve arkadaşları, dağıtık kümelerde veri bloklarının tek bir kümeye taşınıp işlenmesi

verimsiz olduğundan, G-Hadoop adında bir eşle/indirge yapısı tasarlamışlardır. Hadoop'un dosya sistemini Gfarm dosya sistemi ile değiştirmişlerdir. Bununla birlikte eşle/indirge işlerini, dağıtık kümeler üzerinde paralel olarak çalıştırılabileceğini belirtmişlerdir. Kümeler üzerindeki hata toleransını arttırarak eşle/indirge görevlerinin kopyasını farklı kümeler üzerinde oluşturabilmesini sağlamışlardır [37].

Marzuni ve arkadaşları, küme içi ve kümeler arası trafiği ayırdıklarını belirtmişlerdir. Bunu dağıtık Hadoop kümelerinde eşle/indirge programlama modelini çalıştırdıktan (küme içi trafik), Gshuffling adını verdikleri yöntemle anahtar/değer çiftlerinin sadece bulunduğu kümeler arasındaki karıştırılacağı bildirmişlerdir. Ayrıca kümeler arasında anahtar/değer dağılımlarının dengeli olabilmesi için GRG (Global Reduction Graph) adını verdikleri graf yapısını kullanmışlardır. Böylelikle kümeler arasındaki veri hacmini azaltmışlardır [13].

Yapılan çalışmalarda internet üzerinden, veri merkezleri arasındaki veri transferi verimli değildir. Veri merkezlerinde bulunan düğümlerin performansları ve hesaplama düğümü sayısının farklı olabileceği, dağıtık veri merkezlerinde depolanan verilerin boyutu, işleri tamamlama zamanları göz ardı edilmiştir.

Bu çalışmada, dağıtık veri merkezlerindeki verileri eşle/indirge ile işlenebilmesi için bir yöntem önerilmektedir. Önerilen yöntemin aşamalarında; veri işlemeye devam eden veri kümelerinin iş ilerleme durumları kontrol edilmektedir. İş ilerleme durumlarına göre, kümelerin durumları belirlenmektedir. Veri kümelerinin birim zamandaki işlediği veriler, hesaplama parametreleri dahil edilmiştir. Anahtarların dağılımının hesaplaması için kümelere göre anahtarlar gruplanır. Böylelikle anahtar/değer çiftlerinin dengeli dağılımı sağlanmış olur.

2. BÜYÜK VERİ

Gartner, birden fazla veri kaynağından elde edilen çok büyük miktarda verinin toplanarak detaylandırılması için büyük veri terimi önermiştir [38], [39]. Teknolojinin ilerlemesiyle giderek artan dijitalleşme, günlük hayatımızın ayrılmaz bir parçası olmuştur. Dijitalleşmeyle, farklı kaynaklardan (sağlık, devlet, sosyal ağlar, pazarlama, finans vb.) her gün daha önce görülmemiş oranda büyük veri hacimleri üretilmektedir. Bu durum, en çok nesnelerin interneti, bulut bilişim [40] ve akıllı cihazların yaygınlaşması gibi teknolojik eğilimlerden kaynaklanmaktadır.

Büyük veri çağından önce şirketler, tüm arşivleri uzun süreler boyunca saklayamıyor ve çok büyük veri setlerini verimli bir şekilde yönetemiyordu. Çünkü geleneksel teknolojiler sınırlı depolama kapasitesine sahiptir ve pahalıdır. Ayrıca geleneksel teknolojiler büyük veri için ihtiyaç duyulan ölçeklenebilirlik, esneklik ve performanstan yoksundur. Büyük veri yönetimi; önemli kaynaklar, yeni yöntemler ve güçlü teknolojilere ihtiyaç duyar. Daha açık bir ifadeyle, büyük veri ham olduğundan, temizlenmeyi, işlenmeyi, analiz edilmeyi gerektirir. Büyük veriye ihtiyaç duyan kuruluşlar, veri analizinin; rekabetçi olmak, yeni bilgiler keşfetmek ve hizmetleri kişiselleştirmek gibi amaçlarla öneminin giderek arttığının farkındadırlar. Bu nedenlerle dünya genelindeki farklı büyük veri projelerinin bir sonucu olarak, daha fazla depolama kapasitesi, paralel işleme ve farklı kaynaklardan gelen verinin gerçek zamanlı analizini sağlamak için birçok büyük veri modeli çerçevesi oluşturulmuştur. Ayrıca, veri gizliliği ve güvenliğini sağlamak için yeni çözümler geliştirilmiştir. Geleneksel teknolojilerle karşılaştırıldığında, bu tür çözümler daha fazla esneklik, ölçeklenebilirlik ve performans sunmaktadır [41], [42].

Büyük veriyi tanımlayan 3 temel bileşen vardır. Bunlar hacim (volume), hız (velocity), çeşitlilik (variety)'tir [43]. Bazı kaynaklarda ise gerçeklik (veracity) ve değer (value) kavramları da büyük verinin bileşeni olarak kabul edilmektedir. Bileşenler, kendini oluşturan kelimelerin baş harflerinden ötürü 5V olarak bilinmektedir [44], [45].

-Çeşitlilik: Farklı türdeki kaynaklardan gelen veriler; yapısal, yapısal olmayan ve yarı yapısal veriler olarak farklı türlerde olabilir. Verilerin %95'ini yapısal olmayan veriler oluşturmaktadır [44]. Yapılandırılmamış verinin yapısal veri tabanlarında depolanması

mümkün değildir. Örneğin; sensörlerden gelen veriler, günlükler, belgeler her biri farklı türdeki verilerdir.

-Hız: Günümüzde birçok verinin gerçek zamanlı olarak analiz edilmesi gerekebilir. Bu nedenle sürekli üretilen verilerinin işlenme hızının da aynı şekilde artması gerekmektedir.

-Hacim: Büyük verilerdeki bilgiyi işleme becerisinden elde edilen fayda, büyük veri analitiğinin dikkat çekici özelliğidir. Daha fazla veriye sahip olmak daha iyi modellere sahip olmaktan daha iyidir [46]. Sonuç olarak, birçok şirket için çok büyük miktarda çeşitli türde veri depolamak (sosyal ağ verileri, sağlık verileri, finansal veriler, biyokimya ve genetik veriler, astronomik veriler vb.[47]) bir eğilim haline gelmiştir.

-Gerçeklik: Büyük verilerin doğruluğu ve güvenilirliği, verilerin iş kararlarında etkili bir şekilde kullanılabilmesi için kritik öneme sahiptir. Verilerin çeşitliliği, kalite ve güvenilirlik değerlendirme süreçlerini zorlaştırırken, veri kalitesi, güvenilir modellerin üretilmesi için temel bir unsurdur. Verilerdeki aykırı değerler, eksiklikler veya gürültü gibi anomaliler, analizlerin güvenilirliğini azaltabilir. Özellikle sosyal medya gibi kaynaklardan elde edilen verilerdeki belirsizlikler, veri analizlerinde dikkatle ele alınmalıdır. Verilerin doğruluğu ve geçerliliği, yanlış yorumlamalardan kaçınmak için hayati önem taşır. Bu nedenle, veri arındırma, doğrulama ve çapraz denetim gibi süreçler, büyük veri setlerinin yönetiminde önemli rol oynar [48], [49], [50].

-Değer: Büyük verinin değerini ifade eder. Elde edilen ham verinin işlenerek yararlı sonuçlar çıkarılmasıdır.

2.1. HADOOP

Günümüzde veri boyutunun çok yüksek hızla artmasına rağmen, sürücülerin depolama kapasiteleri de büyük ölçüde artmış olsa da, bu sürücülerden veri okuma hızı kayda değer bir gelişme göstermemiştir. Okuma işlemi büyük miktarda zaman alır ve yazma işlemi de daha yavaştır. Bu süre, aynı anda birden fazla diskten okuma yapılarak azaltılabilir. Bir diskin yalnızca yüzde birini kullanmak savurganlık gibi görünebilir. Ancak her biri bir terabayt olan yüz veri kümesi varsa ve bunlara ortak erişim sağlamak da bir çözümdür. Birçok disk arıza olasılığını artırdığı için çok sayıda kullanılmasında da birçok sorun ortaya çıkar. Bu durum çoğaltma ile önlenebilir, yani aynı verinin farklı cihazlarda yedek kopyalarının oluşturulması, böylece arıza durumunda verinin kopyasının kullanılabilir olması sağlanır. Asıl sorun, farklı cihazlardan okunan verilerin birleştirilmesidir. Dağıtık

bilgi işlemde bu sorunun üstesinden gelmek için birçok yöntem mevcuttur ancak yine de oldukça zordur. Tüm bu sorunlar Hadoop tarafından kolayca ele alınmaktadır [51].

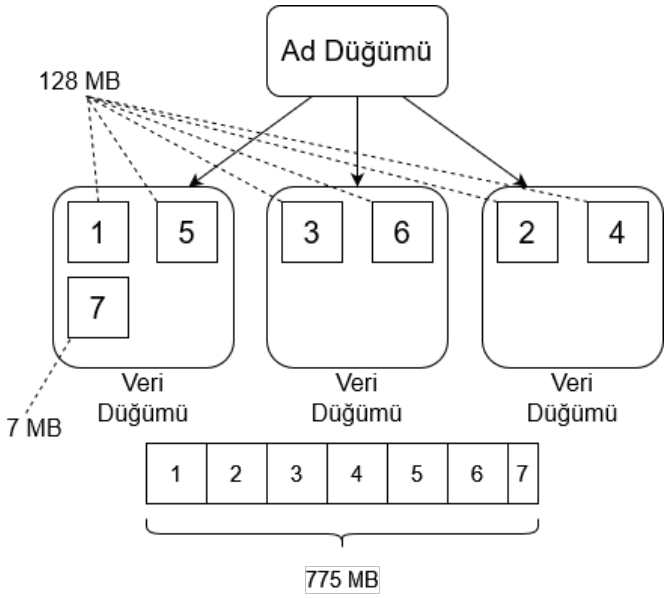
Hadoop büyük verilerin dağıtık olarak bilgisayar kümeleri üzerinde depolanmasını ve işlenmesini sağlayan Java tabanlı açık kaynak kodlu bir projedir. Büyük bir veri dosyasının, bir kümenin düğümlerinde dağıtılmış bir şekilde depolanıp yönetilmesini sağladığından, böl ve yönet stratejisine dayanan bir yazılımdır [52], [53], [54].

Hadoop'un süreci Apache Nutch projesiyle başladı. 2002 yılında Doug Cutting ve Mike Cafarella Apache Nutch projesinde çalışmaya başladılar. Apache Nutch 1 milyar web sayfasını indeksleyebilen bir arama motoru geliştirme süreciydi. Böyle bir sistemin donanım ve işletme maliyeti çok yüksek olmaktaydı. Ayrıca milyarlarca sayfanın bir sistemde işlenmesi verimli olmuyordu. Büyük verilerin depolanması, işlenmesi ve maliyet sorununun çözebilecek bir uygulama arayışı oluştu. 2003 yılında Google tarafından yayınlanmış GFS dosya sisteminin mimarisinin makalesine rastladılar. 2004 yılında Google, eşle/indirge ile ilgili bir makale daha yayınladı. Bu makalelerden yola çıkarak Apache Nutch projesinde Google'ın bu tekniklerini açık kaynak olarak uygulamaya başladılar. Projeye isim olarak Doug Cutting'in oğlunun sarı oyuncak filinin adı olan Hadoop'u verdiler. Proje iki kişinin geliştirebileceğinden çok daha büyük olduğundan Yahoo'ya katılarak geliştirmeye devam ettiler. 2007 yılında proje Apache Software Foundation bünyesinde geliştirilmeye başlandı [55].

Hadoop; HDFS (Hadoop Dağıtık Dosya Sistemi), Eşle/İndirge ve YARN (Yet Another Resource Negotiator) olmak üzere 3 temel bileşenden oluşur.

2.1.1. HDFS

Düşük maliyetli donanımlar üzerinde çalışabilen ve oluşabilecek hatalara karşı dayanıklılık sunan dağıtılmış bir dosya sistemidir. Kullanıcıların ve uygulamaların dosya ve dizin oluşturmasına olanak tanıyan, geleneksel hiyerarşik dosya organizasyonunu destekler. Dosyalar oluşturulabilir, silinebilir, farklı dizinlere taşınabilir veya yeniden adlandırılabilir. Ayrıca, kullanıcı kotaları ve erişim izinlerini de yönetilebilir [56]. Büyük veri kümeleri için tasarlanmış olan HDFS, akış bazlı veri erişimine imkân tanır. Başlangıçta Apache Nutch web arama motoru için geliştirilmiş olan HDFS, Apache Hadoop Core projesinin temel bir bileşenidir ve büyük veri uygulamalarına yüksek verimlilikte erişim sağlar. HDFS'nin temel mimarisi Şekil 2.1'de sunulmuştur.



Şekil 2.1. HDFS mimarisi.

HDFS, ad düğümü, veri düğümü, HDFS istemcisi, görüntü ve günlükler, kontrol noktası düğümü, yedek düğüm ve anlık görüntü bileşenlerinden oluşur.

2.1.1.1. Ad Düğümü (Name Node)

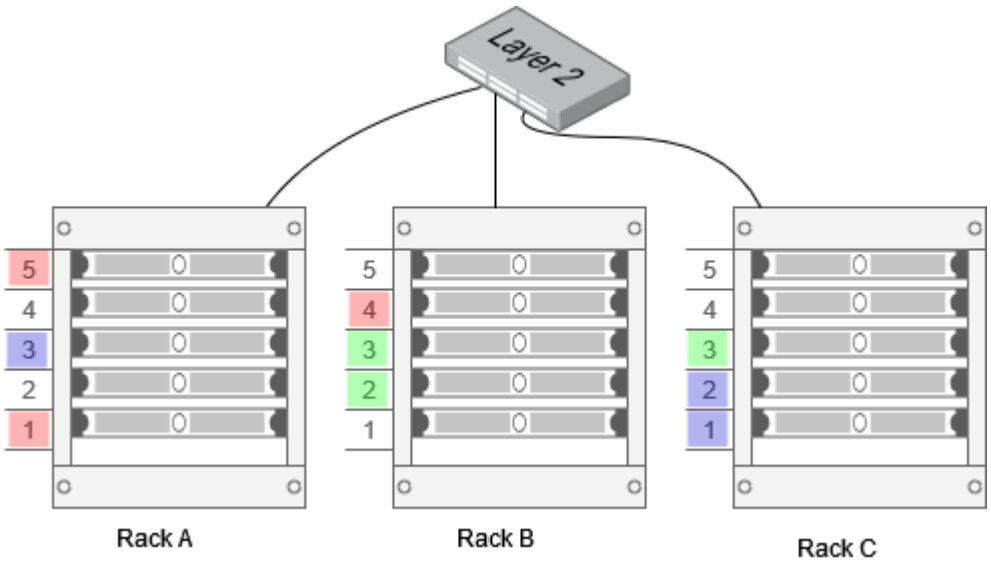
HDFS, dosyaları ve dizinleri bir hiyerarşi içinde yönetir ve bu yapı ad düğümleri üzerindeki inode'lar aracılığıyla temsil edilir. Dosyalar, genellikle 128 MB büyüklüğünde olan bloklara bölünür ve bu bloklar, genellikle üç kopya halinde farklı veri düğümlerine çoğaltılır. HDFS'ye kaydedilen verilerin son blokları dışında, diğer tüm bloklar aynı boyuttadır. Ad düğümleri, dosya bloklarının yerlerini ve ad alanı ağacını saklar. Bir istemci bir dosyayı okumak istediğinde, önce ad düğümünden blok konumlarını alır ve sonra bu verileri en yakın veri düğümünden okur. Yazma işlemleri sırasında, istemci önce ad düğümünden veri yazılacak veri düğümlerini alır ve ardından verileri bu düğümlere yazar. HDFS ad alanı ve dosya sistemini düzenlemek için RAM üzerinde tutulur ve sistem meta verileri, bir kontrol noktası ve günlük dosyasında saklanır. Bu mekanizma, HDFS'nin dayanıklılığını ve veri tutarlılığını sağlar. Ayrıca sistem yeniden başlatıldığında, ad alanının hızlı bir şekilde kurtarılmasına olanak tanır [57].

2.1.1.2. Veri Düğümü (Data Node)

HDFS mimarisinde, her veri düğümü üzerindeki her blok kopyası; veri dosyası ve ona eşlik eden meta veri dosyası olmak üzere iki dosya ile temsil edilir. Veri düğümleri sistem başlatılırken ad düğümü ile iletişim kurar ve ad alanı kimliği ve veri düğümünün sürümlerinin uyumunu doğrular. Eğer uyumsuzluk tespit edilirse, veri düğümleri

otomatik olarak kapanır. Bir veri düğümü, kümeye kaydolduktan sonra düzenli aralıklarla blok raporlarını ad düğümüne gönderir ve bu raporlar; ad düğümlerinin, veri bloklarının nerede saklandığını bilmesini sağlar. Ayrıca, veri düğümleri düzenli kalp atışları göndererek çalışır durumda olduklarını ve depoladıkları verilerin kullanılabilir olduğunu bildirir. Ad düğümleri, bu kalp atışlarını ve blok raporlarını kullanarak kümeyi yönetir, depolama alanı tahsisi ve yük dengeleme gibi işlemler için gerekli kararları alır. Ad düğümünün, kalp atışları üzerinden veri düğümlerine çeşitli komutlar gönderme yeteneği, sistemin sağlıklı ve verimli bir şekilde çalışmasını sağlar [57]. Eğer bir veri düğümünden kalp atışı sinyali alınmazsa o düğüm üzerindeki veriler yok sayılır. Verinin kopyası başka bir veri düğümünde varsa varsayılan çoğaltma sayısı kadar diğer düğümlere çoğaltılır.

HDFS'de depolanan veri bloklarının çoğaltılmasının kuralı şöyledir (Şekil 2.2): HDFS istemcisinin, veri düğümleri kümesinin dışında bulunduğunu varsayalım. İlk kopyayı rastgele bir düğüme yerleştirilir. Ardından, bir sonraki kopyayı farklı bir rafa yerleştirilir. Son olarak, üçüncü kopya ikinciyle aynı raftaki başka bir düğüme yerleştirilir [58]. Üçten fazla çoğaltma varsa diğer kopyalar rastgele düğümlere yerleştirilir. HDFS'deki varsayılan çoğaltma konumlandırma ilkesi, blokları veri düğümleri arasında eşit olarak dağıtmaz, bu da girdi ve çıktılarda tutarsızlıklara neden olur [59].



Şekil 2.2. HDFS blok çoğaltma mimarisi.

HDFS'deki varsayılan 3 kopya çoğaltma tekniği de %200 daha fazla depolama alanı, CPU kullanımı ve ağ bant genişliği gibi ek sistem kaynaklara ihtiyaç duyar [60]. Buna rağmen, çoğaltmalar yalnızca arıza durumunda kullanılır ve gerektiğinde kullanılabilir olmalıdırlar [61].

2.1.1.3. HDFS İstemcisi (HDFS Client)

HDFS istemcisi, kullanıcıların DFS arayüzü üzerinden dosya sistemine erişimini sağlayan bir kütüphanedir. HDFS, okuma, yazma ve silme gibi temel dosya işlemlerinin yanı sıra dizin oluşturma ve silmeyi de destekler. Kullanıcılar, dosyalara ve dizinlere ad alanındaki yollar aracılığıyla erişir [62], [63].

2.1.1.4. Görüntü ve Günlük (Image ve Journal)

Dosya ve dizinlerin, dosya sistemi hiyerarşisindeki meta verileri, görüntüler olarak adlandırılır. Bu görüntünün diske yazılan kalıcı versiyonuna kontrol noktası adı verilir [62], [63].

2.1.1.5. Kontrol Noktası Düğümü (Checkpoint Node)

Düzenli aralıklarla mevcut kontrol noktasını ve günlüğü birleştirerek yeni bir kontrol noktası ve boş bir günlük oluşturur. Bellek ihtiyacı ad düğümü ile aynı olduğundan, genellikle farklı bir sunucuda çalışır [62], [63].

2.1.1.6. Yedek Düğümü (Backup Node)

Periyodik kontrol noktaları oluşturur. Ad düğümünün anlık durumuyla sürekli senkronize olan dosya sistemi ad alanının güncel ve bellekte tutulan bir görüntüsünü saklar.

2.1.1.7. Dosya Sistemi Anlık Görüntüleri (File System Snapshot)

HDFS'de anlık görüntü oluşturma, özellikle güncellemeler sırasında veri kaybını veya bozulmayı önlemek amacıyla kullanılır. Bu mekanizma, sistem yöneticilerine dosya sisteminin bir zaman noktasındaki durumunu kalıcı olarak kaydetme imkânı tanır. Böylece, bir güncelleme işlemi sırasında veri kaybı veya bozulma yaşanırsa, sistem anlık görüntü alındığı zamanki durumuna geri döndürülebilir [62], [63].

2.1.2. Eşle/İndirge

Büyük ölçekli veri işlemeye yönelik artan ihtiyaç nedeniyle, verinin dağıtık olarak paralel ve hızlı bir şekilde işleyebilen bir çerçeve gerekmekteydi. Eşle/indirge, büyük miktarda veriyi dağıtık bir şekilde paralel olarak işlemek için 2004 yılında Google tarafından tanıtılan bir yazılım çerçevesidir [64]. MapReduce çerçevesinin basitliği, büyük ölçekli verilerin eşle ve indirge sınıflarından oluşan fonksiyonlar ile işlenmesinde yatmaktadır.

Bir eşle/indirge programcısı yalnızca bu fonksiyonların mantığına odaklanarak kodlama yapar (Şekil 2.3).

```
map(String anahtar, String değer):
        // anahtar: satırlar
        // değer: kelime
        for each word w in değer:
                cikti(w, "1");
reduce(String anahtar, Iterator
degerler):
        // anahtar: kelime
        // değer: kelime sayisi
        int toplam = 0;
        for each v in değerler:
                toplam += ParseInt(v);
                cikti(AsString(result));
```

Şekil 2.3. Kelime sayma örneğinin eşle/indirge sözde kodu.

Görev paralelleştirme, yük dengeleme, hata toleransı ve veri dağıtımı gibi karmaşık işlemler eşle/indirge çerçevesi tarafından gerçekleştirilir. Bir eşle/indirge çerçevesi kodlandıktan sonra, çerçeve; bu işlevlerin birden fazla örneğini, hesaplama düğümleri kümesinde paralel olarak çalıştırır. Bir eşle/indirge kümesi binlerce hesaplama düğümü içerebilir. HDFS'de depolanan veriler, işlenmek üzere eşle/indirge arayüzüne gönderilir. Eşle/indirge arayüzünde iş, eşle ve indirge olmak üzere iki ana aşamada işlenir. Eşle tamamlandıktan sonra, iş karıştırılır ve bölümlere ayrılır ve ardından daha fazla işlem için indirgeyiciye gönderilir [65], [66].

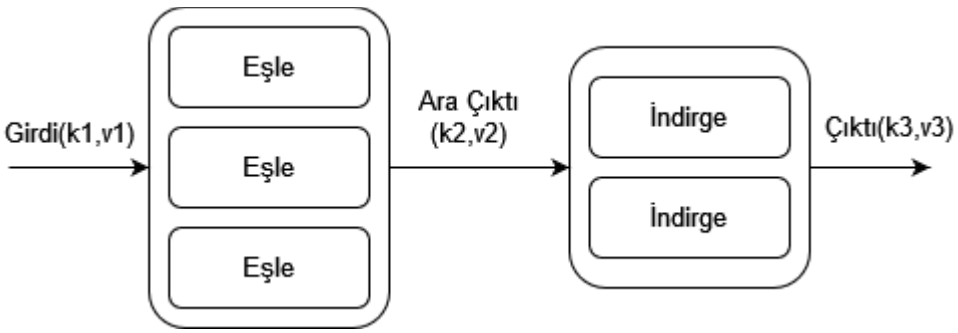
Eşle/indirge programlama modelinin çalışma adımları aşağıda verilmiştir [66]:

1. Programlanan eşle/indirge çerçevesi, kümede tanımlı olan işçi düğümlere, eşle/indirge programını kopyalar.
2. İşçi düğümler üzerinde eşle/indirge programı başlatılır.
3. Her işçi düğümü üzerindeki veri parçaları, kayıt okuyucusu (Record Reader) ile anahtar/değer çiftine dönüştürülür. Varsayılan kayıt okuyucusu 'Text Input Format' sınıfıdır. Varsayılan kayıt okuyucusu için, veri bloklarındaki her satırın konumunun değeri anahtar, satırdaki veriler ise değer olarak eşle görevlerine girdi olarak taşınır.
4. Eşle fonksiyonundaki kodlamaya göre; birden fazla paralel göreve girdi olan anahtar/değer çiftleri işlenerek, yeni anahtar/değer çiftleri oluşturulur.

Anahtar/değer çiftleri tampon belleğe yazılır.

5. Tampon belleğin içeriği periyodik olarak diske yazılır.
6. Tampon bellekteki veriler indirge görevlerinin çalışacağı düğüm sayısına göre ayrılır.
7. Tampon belleklerde depolanan eşle görevlerinin çıktılarındaki aynı anahtarlara ait anahtar/değer çiftleri, aynı indirge görevinin çalışacağı düğümlere taşınarak karıştırma gerçekleşir. Karıştırma, indirgenin çalışacağı düğüm tarafından; HTTP GET ile eşle görevlerinin çıktısının olduğu yerel diskten gerçekleşir [67]. İndirge görevlerinin çalıştırılmasına başlamadan önce tüm eşle görevlerinin süreçlerinin tamamlanması gerekir.
8. İndirge görevlerinin çalışacağı düğümlere taşınan aynı anahtarlar gruplanır ve sıralanır.
9. Kullanıcının belirlediği indirge görevlerinde her anahtar yinelemeli olarak işlenir. Göreve girdi olarak verilen anahtar/değer çiftlerinden, çıktı olarak yeni anahtar/değer çiftleri üretilir.
10. Üretilen anahtar/değer çiftleri çıktı dosyalarında toplanır.

İndirge fonksiyonlarının işlerini tamamladıktan sonra eşle/indirge işi tamamlanır. Şekil 2.4'te eşle/indirge programlama modelinin mimarisi verilmiştir.

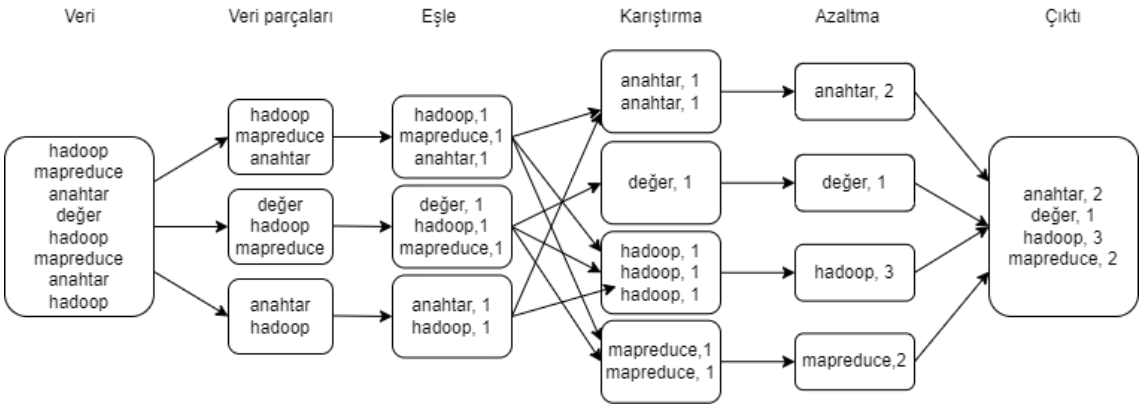


Şekil 2.4. Eşle/indirge programlama modelinin mimarisi.

HDFS'de depolanan veriler üzerinde çalıştırılan; kelime sayma eşle/indirge işi Şekil 2.5'te, iz kayıtlarını filtreleyen eşle/indirge işi ise Şekil 2.6'da verilmiştir.

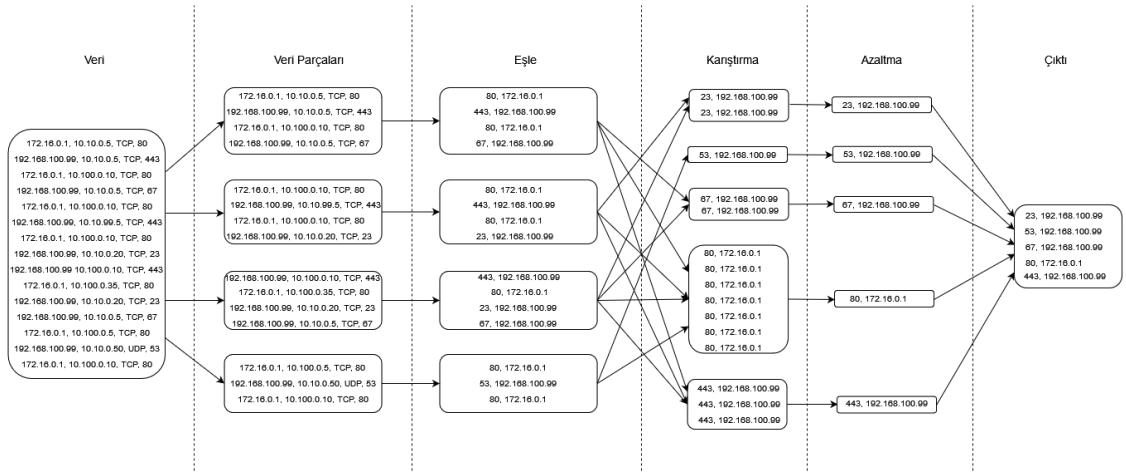
HDFS'de depolanacak veriler, veri bloğu boyutuna göre veri parçacıklarına ayrılır. Bu parçacıklara veri bloğu adı verilir. Her veri bloğu üzerinde bir eşle görevi çalıştırılır. Kelime sayma örneği için, eşle görevinde işlenen her veri bloğundaki verilerden, her

kelime için anahtar, değer (kelime,1) olarak çıktı üretilir. Eşle görevinin çıktılarındaki aynı anahtarlar karıştırma aşamasıyla birlikte gruplanır ve sıralanır. İndirge görevinde ise aynı anahtarların değerleri toplanıp kelimelerin sayıları belirlenerek çıktı üretilir.



Şekil 2.5. Kelime sayma eşle/indirge örneği.

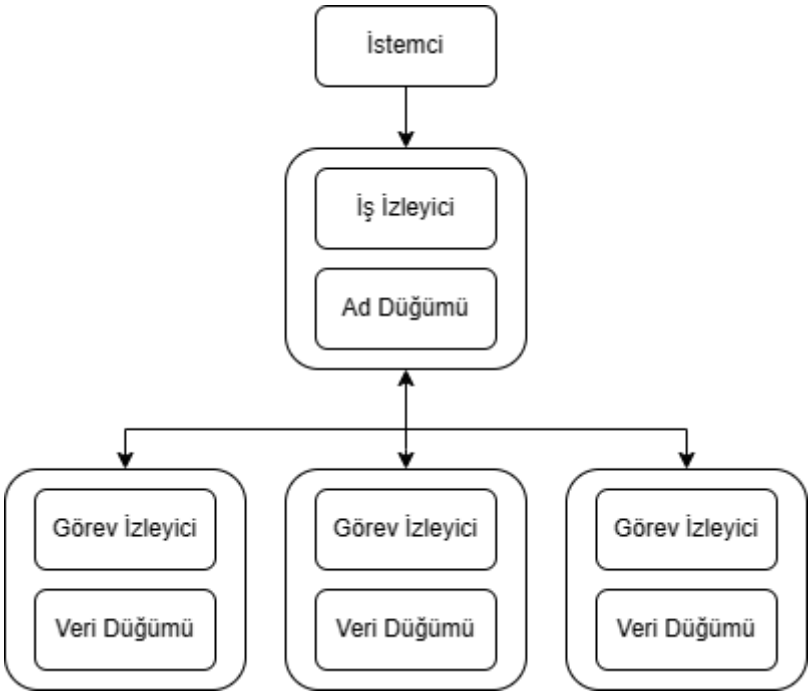
İz kayıtları örneği için, bir şirketin internet trafiği iz kayıtlarında kaynak IP adresi, hedef IP adresi ve hedef port numarası saklanmaktadır. İz kayıtlarından servislerin hangi IP adresleri tarafından kullanıldığı bir eşle/indirge işi ile belirlenir. Bu örnek için servislere karşılık hedef port numaraları anahtar olarak belirlenir. İz kayıtları HDFS'de, veri parçaları olarak depolanır. Veri parçalarında eşle fonksiyonu çalıştırıldıktan sonra, çıktısı olarak anahtar/değer çiftleri (port numarası, kaynak IP adresi) belirlenir. Aynı port numarasına sahip kaynak IP adresleri karıştırma aşamasıyla port numarasına göre gruplanır ve sıralanır. İndirge görevinde hedef port numaralarına göre kaynak IP adresleri birleştirilir. Aynı kaynak IP adresleri göz ardı edilir. Farklı kaynak IP adresleri birleştirilerek indirge görevinin çıktıları (80,192.168.100.66) üretilir.



Şekil 2.6. İz kayıtları eşle/indirge örneği.

Eşle/indirge, olağanüstü hata toleransı yetenekleri, ölçeklenebilirliği ve kullanım kolaylığı nedeniyle biyoinformatikte çoklu dizi hizalama [68], dağıtılmış veri sıralama, dağıtılmış arama/indeksleme, web bağlantılarının grafları tersine çevirme, web erişim günlüğü istatistikleri [69] ve makine öğrenimi [70] dahil olmak üzere oldukça geniş bir uygulama yelpazesinde kullanılır. Genel olarak eşle/indirge, az miktarda veri üzerinde karmaşık hesaplamalar yapmak yerine önemli miktarda veri üzerinde hesaplama yapmak için tasarlanmıştır [71].

Eşle/indirge, MRv1 (Şekil 2.7) ve MRv2 (Şekil 2.8) olarak iki sürümü bulunmaktadır.



Şekil 2.7. MRv1 mimarisi.

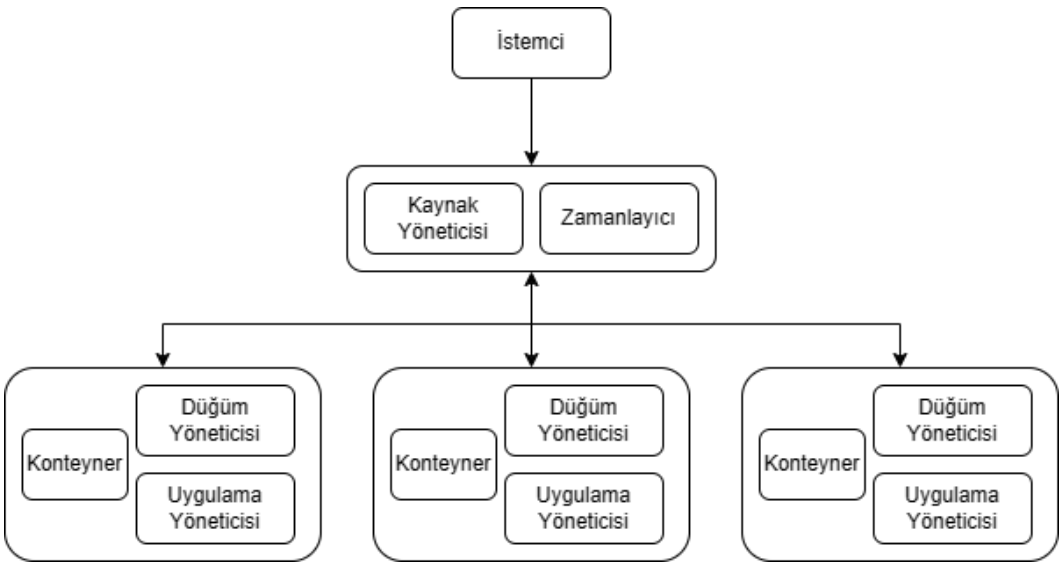
MRv1 olarak adlandırılan ilk sürüm, Hadoop eşle/indirge; temel olarak iki bileşene dayanmaktadır: MRv1 mimarisinde, ad düğümünde iş izleyici, veri düğümlerinde de görev izleyici bulunur. Eşle/indirge çerçevesinde ad düğümü, farklı işleri programlamak ve görevleri bağımlı düğümlere dağıtmak için iş izleyici çalıştırır. İşlerin başarılı bir şekilde çalışmasını sağlamak için iş izleyici, kendisine bağlı işçi düğümlerin durumunu izler ve başarısız olduklarında görevleri yeniden atar. Bağlı işçi düğümlerin her biri, atanan görevler için bir görev izleyici örneği çalıştırır. Bir görev izleyici, görevleri iş izleyici tarafından belirtildiği şekilde yürütür ve yürütülmelerini izler. Her görev izleyici, birkaç eşle veya indirge görevini paralel olarak yürütmek için birden fazla JVM (Java Sanal Makinesi) kullanabilir [42], [72], [73].

2.1.3. YARN (Yet Another Resource Negotiator)

Birden fazla veri işleme uygulaması Hadoop kümeleri üzerinde çalıştığında, kümelerde işlerin koordinasyonunun sağlanması ve kaynak paylaşımı kritik öneme sahiptir. Çoklu hesaplama katmanları arasındaki kaynak paylaşımı, büyük veri işlemenin vazgeçilmez bir özelliğidir.

Hadoop, kaynak paylaşımı ve ölçeklenebilirlik sorunlarını [74] çözmek için, YARN teknolojisini kullanır. Bu teknoloji Hadoop'un 2.0 sürümüyle tanıtılmıştır. MRv1'deki iş izleyici ve görev izleyici yerine kaynak yöneticisi (Resource Manager) ve uygulama yöneticisi (Application Master) gibi bileşenlerle kaynak planlaması ve görevlerin çalıştırılması sağlanarak, Hadoop'un yeteneklerini geliştirir. Ayrıca, düğümlerde çalışan düğüm yöneticisi (Node Manager) servisi, kaynak yönetimi ve durum raporlaması gibi görevleri yerine getirir. Bir uygulama başlatıldığında, o uygulama için özel bir uygulama yöneticisi başlatılır ve kaynak yöneticisi ile düğüm yöneticisi arasında kaynakları etkili bir şekilde yönetir. Bu yapı, Hadoop'un büyük veri işleme kabiliyetlerini önemli ölçüde geliştirmiştir. YARN, kaynak yönetimi, iş zamanlama ve izleme işlevlerini ayrı görevlere böler. YARN, her küme için bir kaynak yöneticisi ve uygulama başına uygulama yöneticisine sahiptir [75].

YARN bileşenlerinden oluşan eşle/indirge veri işleme sürümü MRv2'nin mimarisi Şekil 2.8'de sunulmuştur.



Şekil 2.8. MRv2 mimarisi.

Kaynak yöneticisi, düğüm yöneticisi ve uygulama yöneticisi veri hesaplama çerçevesini oluşturur. Kaynak yöneticisi, sistemdeki bir kümede çalıştırılan uygulamalar arasında kaynak atamasını yapan bileşendir. Düğüm yöneticisi, konteynerlerden sorumludur, CPU, RAM, disk, ağ kullanımlarını izler ve bunları kaynak yöneticisi ve zamanlayıcıya raporlar. Her işçi düğümde bir tane bulunur. Kaynak yöneticisi, zamanlayıcı ve uygulama yöneticisi olmak üzere iki bileşenden oluşur. [76]. Uygulama yöneticisi ise, kaynak yöneticisi tarafından belirlenen kaynakları ayarlamak, görevleri yürütmek ve izlemek için düğüm yöneticisi ile çalışmakla görevlidir.

YARN ile birlikte gelen MRv2'deki en büyük değişikliklerden biri kaynakların yönetilme şeklidir. MRv1'de her düğüm belli sayıda eşle slotu ve indirge slotu ile yapılandırılırken YARN ile, eşle için kullanılabilir kaynaklar ile indirge için kullanılabilir kaynaklar arasında bir kısıtlama yoktur. Tüm kaynaklar her ikisi için de kullanılabilir. Ayrıca slot kavramı yerine kaynaklar, bellek miktarı ve CPU sayılarına göre yapılandırılmaktadır. Böylelikle daha etkin kaynak yönetimi sağlanabilmektedir [77].

2.2. DAĞITIK VERİ MERKEZLERİNDE VERİ İŞLEME MODELLERİ

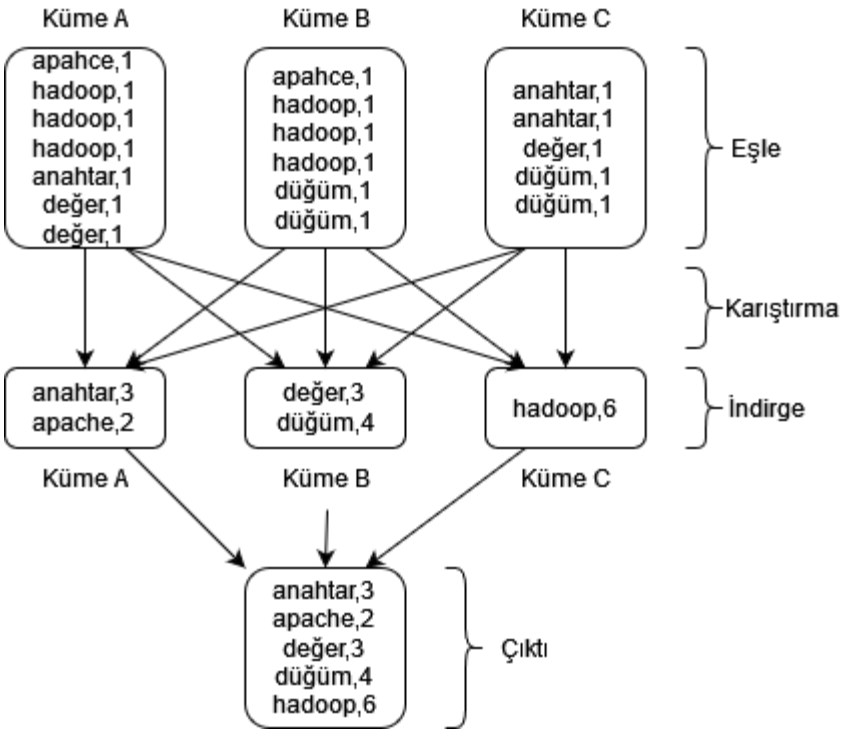
Günümüzde büyük ölçekli pek çok şirket farklı coğrafi konumlarda veri merkezlerine sahip olabilir. Üretilen verilerin boyutu büyük olduğunda veya yasal kısıtlamalar nedeniyle, verilerin üretildiği yerlere yakın veri merkezlerinde depolanması gerekir. Apache Hadoop'un popüler uygulaması MapReduce sunulduğundaki varsayım, veri merkezlerindeki tüm düğümlerin tek kümede ve aynı olmasıdır. Ancak gerçek ortamda, bulut adı verilen uygulamalar için işleme ve depolama düğümleri tarafından sağlanan topolojiler birbirinden farklı olabilir. Mantıksal olarak tek bir veri kümesinden oluşan dağıtık veri merkezleri, fiziksel olarak farklı konumlardaki veri merkezlerinin birleşiminden oluşur [78].

Coğrafi olarak dağıtılmış verileri işlemek için gerçekleştirilebilecek yöntemlerden biri verilerin tek bir veri merkezine kopyalanmasıdır. Bu, depolama ve taşıma maliyetiyle önemli bir dezavantaj oluşturur. Dağıtık veri merkezlerindeki verilerin işlenmesi için Geo-Hadoop ve Hiyerarşik yaklaşım olmak üzere iki temel yaklaşım bulunmaktadır [36].

2.2.1. Geo-Hadoop Yaklaşımları

Geo-Hadoop, verilerin farklı DC'ler üzerinde depolanarak dağıtık olarak işlendiği bir yaklaşımdır. Coğrafi olarak dağıtılmış kümelerdeki düğümlerin performansını ve ağların farklılığını hesaba katan temel Hadoop uygulamasının geliştirilmiş versiyonlarıdır.

Geo-Hadoop temelde iki mimari altında kategorize edilmiştir: Bunlar merkezileştirilmiş mimari ve merkezi olmayan mimaridir [28]. Merkezileştirilmiş mimari, DC'lerin birinde bulunan bir yönetici düğüm, diğer DC düğümlerinin kaynaklarını yönetir. Merkezi olmayan mimaride ise her DC kendi işçi düğümlerini yönetir. Her DC kendi içinde Hadoop işlerini çalıştırabilir. Ayrıca DC'ler coğrafi olarak dağıtılmış işler için hesaplama ve veri kaynağını birbirleriyle paylaşabilir. Geo-Hadoop yaklaşımı için örnek topoloji Şekil 2.9'da sunulmuştur.



Şekil 2.9. Geo-Hadoop yaklaşımı örnek topolojisi.

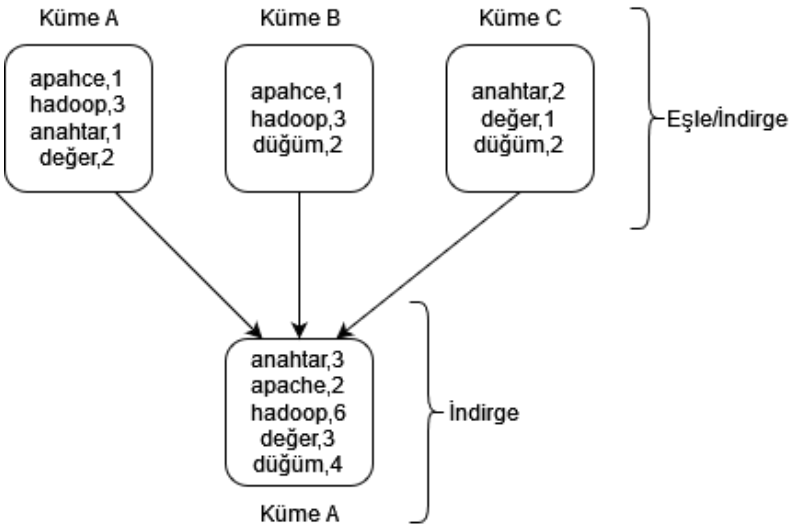
Farklı konumlarda bulunan kümelerin her birinde kelime sayma uygulamasının eşle fonksiyonu çalıştırılarak (key, 1) anahtar, değer çiftleri elde edilmiştir. Aynı anahtara ait çok sayıda değer üretilmiştir. Örneğin Küme A'da, (hadoop, 1) üç tane aynı anahtara ait, (değer, 1) iki tane aynı anahtara ait anahtar/değer çifti bulunmaktadır. Küme B'de, (hadoop, 1) üç tane aynı anahtara ait, (düğüm, 1) iki tane aynı anahtara ait anahtar/değer çifti bulunmaktadır. Küme C'de, (anahtar, 1) iki tane aynı anahtara ait, (düğüm, 1) iki

tane aynı anahtara ait anahtar/değer çifti bulunmaktadır. Eşle görevinden sonraki karıştırma aşamasında bir kümeden birden fazla aynı anahtara ait (anahtar, değer) çifti taşınacaktır. Aynı anahtara ait (anahtar, değer) çiftleri aynı kümelere taşındıktan sonra indirge görevinde tekil (anahtar, değer) çiftleri (anahtar, 3) (apache, 2) (değer, 3) (düğüm, 4) (hadoop,6) üretilerek eşle/indirge işi tamamlanmış olur.

2.2.2. Hiyerarşik-Hadoop Yaklaşımları

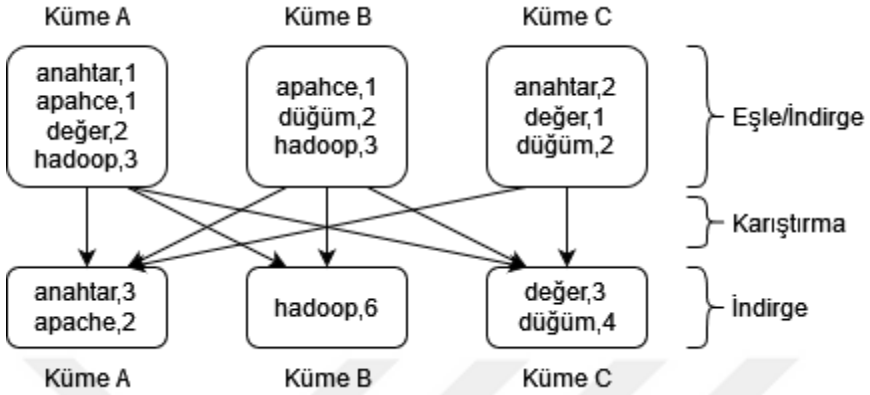
Hiyerarşik hadoop yaklaşımında temel yöntem, Hadoop'un hesaplama yeteneğinden küme bazında yararlanmak ve farklı kümelerdeki dağıtık hesaplamalardan toplanan sonuçları birleştirmektir. Bu yaklaşımda, her veri merkezi bir eşle/indirge işini tamamlar. Kümelerin anahtar/değer çıktıları tek bir küresel indirgeyiciye aktarılır. Küresel indirgeyicide indirge görevi çalıştırılarak nihai çıktı elde edilir. Eşle/indirge-küresel indirgeme yaklaşımı iki katmanlı yapıya sahip olmasına rağmen, ihtiyaca göre daha fazla katmana sahip hiyerarşik yaklaşımlara da uyarlanması mümkündür [79].

Hiyerarşik Hadoop yaklaşımı ile ilgili örnek topoloji Şekil 2.10'de sunulmuştur. Örnekte dağıtık kümelerdeki veriler üzerinde kelime sayma uygulaması çalıştırılmıştır. Kümelerin tümünde, eşle/indirge uygulamasından sonra; Küme A'da (apache, 1) (hadoop, 3), (anahtar ,1), (değer, 2), Küme B'de (apache, 1), (hadoop, 3), (düğüm, 2), Küme C'de (anahtar, 2), (değer, 1), (düğüm, 2) anahtar/değer çiftleri olarak, her kümede tekil anahtarlar üretilir. Bu anahtar/değer çiftleri bir kümede toplanıp, indirge fonksiyonu ile birleştirilerek (anahtar ,3), (apache, 2), (hadoop, 6), (değer, 3), (düğüm, 4) anahtar/değer çiftleri elde edilir.



Şekil 2.10. Hiyerarşik Hadoop yaklaşımı örnek topolojisi.

Literatürde hem Hiyerarşik Hadoop hem de Geo-Hadoop yaklaşımlarının birlikte ele alındığı çalışmalar bulunmaktadır. Hiyerarşik yaklaşımın ilk fazındaki eşle/indirge aşamasından sonra, verilerin Geo-Hadoop yaklaşımının ikinci fazında olduğu gibi anahtar/değer çiftleri çoklu indirgeyicilere gönderilerek, indirge fonksiyonu çalıştırılır.



Şekil 2.11. Hibrit Hadoop yaklaşımı örnek topolojisi.

Hibrit Hadoop yaklaşımları ile ilgili örnek topoloji Şekil 2.11'de sunulmuştur. Dağıtık yapıdaki veri kümeleri üzerinde kelime sayma eşle/indirge uygulaması çalıştırılmıştır. Çıktı olarak Küme A'da (anahtar, 1), (apache, 1), (değer, 2), (hadoop, 3); Küme B'de (apache, 1), (düğüm, 2), (hadoop, 3); Küme C'de (anahtar, 2), (değer, 1), (düğüm, 2) tekil anahtar/değer çiftleri oluşmuştur. Sonraki aşamada farklı kümelerdeki aynı anahtarlar, aynı kümede olmak üzere toplandıktan sonra, indirge işi çalıştırılır. Küme B'den (apache, 1) ve Küme C'den (anahtar, 2) anahtar/değer çiftleri Küme A'ya; Küme A'dan (hadoop, 3) anahtar/değer çifti Küme B'ye; Küme A'dan (değer ,2) ve Küme B'den (düğüm, 2) anahtar/değer çiftleri Küme C'ye karıştırma fazında gönderilir. Sonrasında, indirge aşamasında, kümelerdeki aynı anahtarlar birleştirilir.

2.3. REGRESYON

İki veya daha fazla değişkenin, aralarındaki ilişkiyi saptayarak ulaşılan sonuçları değerlendirip, çıkarımlarda bulunulmasını sağlayan istatistiksel bir yöntemdir. İngiliz istatistikçi Sir Francis Galton tarafından, uzun boylu anne babaların çocuklarının uzun, kısa boylu anne babaların çocuklarının kısa boylu olması beklentilerine karşı, çocukların boylarının ortalama nufüsun, boy uzunluğuna yaklaşması eğiliminde olduğunu gösteren bir çalışma yapmıştır. Böylelikle ilk defa regresyon analizi ortaya çıkarılmıştır [80], [81].

Bir regresyon analizinde amaç bağımsız değişkenler ile bağımlı değişkenler arasında

ilişkiyi bularak, bağımsız değişkenlerden bağımlı değişkenin tahmin edilmesidir. Örneğin, bir gayrimenkul şirketi bir evin değerinin tahmini için; evin metrekaresi, oda sayısı, konumu, evin yaşı, asansörü ve bulunduğu kat ile bağımsız değişkenleri oluşturur. Bu bağımız değişkenlere karşılık evin değeri, bağımlı değişkenler olarak ele alınır. Geçmiş elde edilen bağımlı ve bağımsız değişken verileri arasındaki ilişkisi matematiksel yöntemle ile ifade edilerek regresyon yapılmış olur. Regresyon analizinin amaçları şunlardır [82]:

1. Tahmin: Önceden belirlenmiş denklem kullanılarak bağımsız değişkenlerden bağımlı değişkenin değeri tahmin edilir.

2. İlişkiyi Anlama: Bağımlı değişkenler ile bağımsız değişkenler arasında ilişkinin belirlenmesi amaçlanır. Bağımsız değişkenlerin bağımlı değişken üzerindeki etkisi belirlemek için katsayılar ve istatistiksel veriler kullanılabilir.

3. Değişkenlerin Etkisini Ölçme: Bağımsız değişkenlerin bağımlı değişkenler üzerindeki etkilerinin ölçmek için kullanılır. Bağımsız değişkenlerin katsayılarının bağımlı değişkenler üzerinde ne kadar değişiklik yaptığını gösterir.

4. Tahmin Değerlerinin Güven Aralığını Hesaplama: Regresyon analizi, tahmin edilen değerlerin güven aralığını hesaplamak için kullanılabilir. Bu, tahminlerin ne kadar doğru veya güvenilir olduğunu anlamaya yardımcı olur.

Regresyon analizinde, değişkenlerin doğru bir şekilde tanımlanması önemlidir. Bu analiz, değişkenler arasındaki ilişkiyi ortaya çıkarır, ancak bu ilişkilerin nedenselliğini belirtmez. Regresyon denkleminin en uygun formunu elde etmek için bağımsız değişkenlerin seçimine özen gösterilmesi gerekmektedir. Her bir bağımsız değişkenin bağımlı değişkenle güçlü bir ilişkisi olması ve diğer bağımsız değişkenlerle ilişkisinin olmaması önemlidir. Ayrıca, veri setlerinin yeterli büyüklükte olması, çoklu doğrusal bağımlılığın olmaması ve aykırı değerlerin bulunmaması gibi durumlar da analiz öncesinde kontrol edilmelidir [83]. Uç değerler, veri kümesindeki gözlemlerin çoğuna aykırı olan gözlemleri ifade eder [84]. Uç değerlerin bulunduğu veri kümesinde varsayımlar sağlanamadığından, kurulan regresyon modellerinden oluşan tahminler yanıltıcı olmaktadır. Uç değerlerin veri setinden çıkarılması, regresyon denkleminde önemli değişikliklere neden olabilir. Bu nedenle, büyük artık değerlere sahip gözlemler, regresyon analizinde son derece etkilidir. Bu tür durumlarda, uç değerlerin tanımlanması ve sonuçların güvenilirliği için güçlü regresyon yöntemlerinin tercih edilmesi daha

uygundur. Bu nedenlerden dolayı regresyon analizi öncesinde verilerin incelenmesi oldukça önemlidir [85], [86].

2.3.1. Basit Doğrusal Polinom Regresyon Yöntemi

Tek bir bağımsız değişken ve bağımlı değişkenin kullanıldığı regresyon yöntemidir. Tek değişkenli polinom regresyon eğrisinin denklemi aşağıda şekilde hesaplanır [87], [88], [89]:

$$y^i = \beta_0 + \beta_1 x_i + +\beta_2 x_i^2 + \cdots + \beta_k x_i^k + \varepsilon_i \qquad i = 1,2,\dots,N$$

Bu eşitlik için,

y^i: Bağımlı değişkenin i. gözlem değerini,

x_i: Bağımsız değişkenin i. gözlem değerini,

N: Birim sayısını,

β_0: Regresyon sabitini,

$\beta_0, \beta_1, \dots, \beta_k$: Regresyon katsayılarını,

k: Bağımsız değişkenin derecesini,

ε_i: Modele ait hata terimini göstermektedir.

2.3.2. Çoklu Doğrusal Polinom Regresyon Yöntemi

Çok sayıda bağımsız değişken ve bir bağımlı değişkenin kullanıldığı regresyon yöntemidir. Klinik ve davranışsal bilimlerde yaygın olarak kullanılan istatistiksel yöntemlerden biridir. Çoklu doğrusal regresyon, araştırmacıya birden çok değişkenle çalışabilme ve esnek araştırma alanı oluşturmayı sağlamaktadır [90], [91]. Örneğin bir çiçeğin büyümesini etkileyen faktörlerden toprağın nemi, mineralleri, ışık alma süreleri bağımsız değişken; bitkinin birim zamandaki uzama hızı ise bağımlı değişken olarak değerlendirilir. Aşağıda örnekte çoklu polinom regresyon eşitliği sunulmuştur:

$$y^i = \beta_0 + \beta_1 x_{1i} + +\beta_2 x_{2i}^2 + \cdots + \beta_k x_{ki}^k + \varepsilon_k \qquad i = 1,2,\dots,N$$

Bu eşitlik için,

y^i: Bağımlı değişkenin i. gözlem değerini,

x_{ti}: t bağımsız değişkenin i. gözlem değerini,

N: Birim sayısını,

β_0: Regresyon sabitini,

$\beta_0, \beta_1, \ldots, \beta_k$: Regresyon katsayılarını,

k: Bağımsız değişkenin derecesini,

ε_i: Modele ait hata terimini göstermektedir.

Çoklu regresyon analizinde hangi bağımsız değişkenlerin bağımlı değişkeni daha fazla etkilediği araştırılır [92].

2.3.3. Regresyon Analizlerinde Katsayıların Bulunması

Regresyon analizlerinde en çok kullanılan tahmin yöntemlerinden birisi en küçük kareler (EKK) tahmin yöntemidir. Bu yöntemde regresyon eşitliğindeki parametrelerin tahmini, bağımsız ve bağımlı değişkenlerin oluşturduğu noktalar ile örneklemin regresyon eğrisi denklemine verildiğinde bağımlı değişkenin gözlenen değeri arasındaki hata terimlerini vermektedir. EKK yöntemi, regresyon doğrusu ile tahminler arasındaki sapmaların karelerinin toplamını en küçük yapacak şekilde katsayıların tahmin edilmesini sağlamaktadır. Sapmaların karelerinin alınmasının nedeni, toplam alınırken artı ve eksi değerlerin birbirini götürmesini engelleyerek hatayı elde etmektir [93].

EKK yöntemi yaygın olarak kullanılmaktadır. Bunun nedenleri arasında kullanımının basitliği ve istatistiksel özelliklerinin uygunluğu yer almaktadır. Parametre tahmini, örneklem verilerinden hareketle popülasyona ait bir parametrenin değerini belirlemeyi amaçlar [91].

3. GSELF-MAPREDUCE

Hadoop, dağıtık kümelerde eşle/indirge programlama modeli için herhangi bir arayüz sunmaz. Dağıtık veri merkezlerinde eşle/indirge ile veri işleme için, Geo-Hadoop ve Hiyerarşik Hadoop olarak iki temel yaklaşım bulunmaktadır. Bu iki yaklaşımın yanısıra bu yaklaşımlardan oluşan hibrit yaklaşımlarla da, eşle/indirge işi çalıştırılarak, veri işleme gerçekleştirilebilir.

Bu çalışmada dağıtık kümelerdeki verilerin eşle/indirge ile işlenebilmesi için bir yöntem önerilmektedir. Önerilen yöntem iki aşamadan oluşmaktadır. Öncelikle tüm kümeler üzerinde eşle/indirge işi başlatılır. İlk aşamada işini bitiren kümelerin birim zamandaki işlediği veri istatistikleri göre, veri işleme maliyetleri belirlenir. İkinci aşamada, anahtar dağılımı yapılarak kendi aralarında karıştırmayı gerçekleştirirler.

Önerilen yöntem GSelf-MapReduce olarak adlandırılmıştır. Bu yöntemde kümeler veri işleme sürecini bitirdiklerinde, kendi aralarında karıştırma yapmaları sağlanır. Böylelikle kümelerin hepsinin işini bitirmesi beklenmez. Çünkü kümeler üzerindeki verilerin boyutu ve kümelerin hesaplama kapasiteleri farklı olabileceğinden, eşle/indirge fonksiyonun çalışma süreleri de birbirinden farklı olabilir.

Önerilen GSelf-MapReduce yönteminin test ortamı için referans alınan fiziksel ortamda, veri merkezi geçidi (DCG), veri merkezi yöneticisi (DCM) ve kümelerden oluşan bir yapı oluşturulmuştur. Kümeler ve DCG farklı coğrafi konumlardadır. Her DC'de bir küme bulunmaktadır. GSelf-MapReduce'un çalışma adımları Şekil 3.1'te verilmiştir. Buna göre;

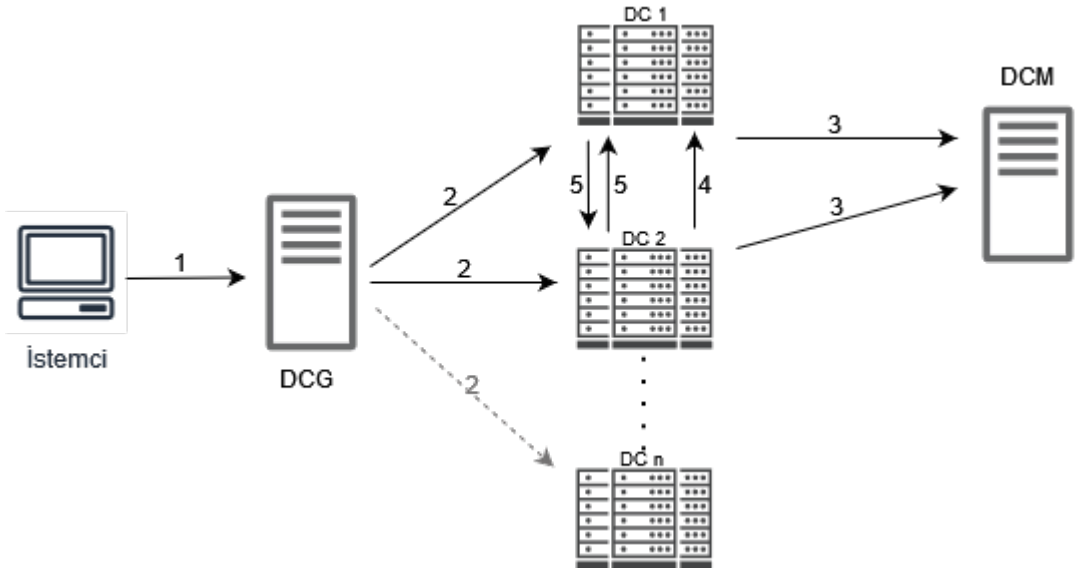
1. adım: İstemci DCG'ye eşle/indirge işini gönderir (1). DCG de kendisinde tanımlı olan kümelere bu işi gönderir (2).

2. adım: Kümeler üzerinde eşle/indirge işi çalıştırılır. Eşle/indirge işini tamamlayan kümeler DCM'ye bildirim gönderir. (3).

3. adım: Anahtar boyutu küçük olan küme, anahtar boyutu büyük olan kümeye anahtarları gönderir (4).

4. adım: DCM tarafından, kümelerin veri işleme maliyeti hesaplanır.

5. adım: Kümelerin maliyetlerine göre anahtarların toplandığı küme üzerinde, kümeler arasındaki anahtar dağılımı gerçekleştirilir. Anahtar dağılımları kümelere gönderilir.

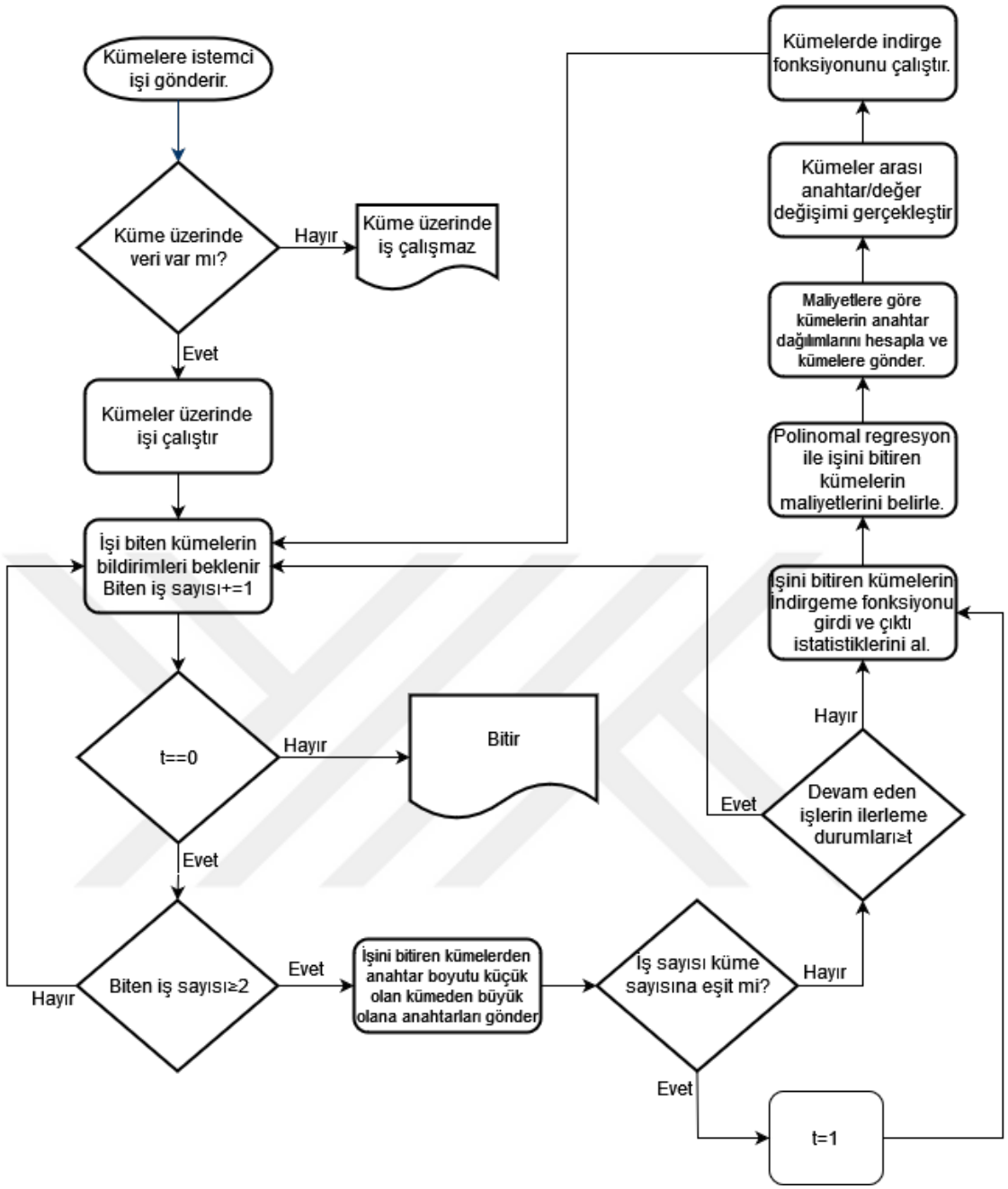


Şekil 3.1. GSelf-MapReduce yönteminin adımları.

6. adım: Kümelerdeki anahtar değer çiftleri, ait olduğu veri merkezlerine gönderilerek, kümeler kendi aralarında karıştırma işlemini gerçekleştirir (5).

7. adım: Karıştırma işlemi tamamlandıktan sonra indirge işi çalıştırılır.

Tüm kümeler işlerini tamamlayana kadar, kümeler arası karıştırma ve indirge süreçleri (3-7 adımları) devam eder. Tüm kümeler aynı anda karıştırma ve indirge aşamalarını gerçekleştirdiğinde veri işleme tamamlanır. Önerilen yöntemin akış diyagramı Şekil 3.2'de verilmiştir.



Şekil 3.2. GSelf-MapReduce yönteminin akış diyagramı.

3.1. KÜMELER ARASI İNDİRGEME

Şekil 3.2'te verilen akış diyagramına göre, karıştırma gerçekleştirecek kümeler, veri işleme ilerleme durumlarına göre belirlenir. Kümelerin indirge görevini çalıştırmadaki performansı değerlendirilerek, kümelerin maliyetleri oluşturulur. Burada oluşturulan maliyet, veri işleme için geçen süre olarak saniye cinsinden belirlenmiştir. Bu aşamada kullanılan parametreler şöyle verilmiştir:

I = İşini bitiren tüm kümeler,

J = Devam eden tüm kümeler,

DC_{cmp} = İşini bitiren kümelerin sayısı,

DC_i = İşini bitiren kümeler

P_i = Devam eden kümelerin ilerleme durumu,

Dk_i = Kümelerdeki çıktı anahtarların veri boyutu,

SRi_i = İndirge girdi veri boyutu,

SRo_i = İndirge çıktı veri boyutu,

CRi_i = İndirge girdi veri sayısı,

CRo_i = İndirge çıktı veri sayısı,

Wi_i = İndirge girdi ağırlığı,

Wo_i = İndirge çıktı ağırlığı,

Verilerin bulunduğu kümelerin sayısına göre farklı senaryolar ele alınabilir. Verilerin 4 farklı küme üzerinde olduğu bir senaryoda; eşle/indirge veya indirge işini başarıyla bitiren DC_i'den DCM'ye işin kimliği (jobID) ile bir bildirim gönderilir. İşini bitiren küme sayısı 1 arttırılır. Bunlar Denklem 3.2 ve 3.3'te sunulmuştur:

$$DC_{cmp} = 0 \tag{3.1}$$

$$DC_i \xrightarrow{jobID} DCM \quad \forall i \in I \tag{3.2}$$

$$DC_{cmp} = DC_{cmp} + 1 \tag{3.3}$$

DCM'nin aldığı bildirim sayısı iki olursa Denklem 3.4,

$$DC_{cmp} \geq 2 \tag{3.4}$$

İşini bitiren kümeler için, çıktı boyutu küçük olan kümelerden (3.5), büyük olan kümeye anahtarlar gönderilir. (3.6). Burada belirtilen işlemler için aşağıdaki denklemler kullanılmıştır.

$$Dk_i \geq Dk_{i+1} \tag{3.5}$$

$$DC_{i+1} \xrightarrow{anahtar} DC_i \tag{3.6}$$

Sonraki aşamada veri işleme işi devam eden DC'lerdeki kümelerin eşle/indirge işlerinin ilerleme durumu kontrol edilir ve belirlenen bir ön tanımlı değerden (t- veri işleme işleminin tamamlanma yüzdesi) büyük olursa, işini bitiren kümelerden yeni bir bildirim beklenir (3.7). Son bildirim gönderen kümelerin anahtarlarının çıktı boyutu DC_i üzerindeki anahtarların boyutundan küçükse (3.8), DC_{i+2}'den DC_i'ye anahtarlar taşınır. (3.9)

$$P_j > t \quad \forall j \in J, 0 \leq P_j \leq 100 \tag{3.7}$$

$$\sum_{i=1}^{DC_{cmp}-1} Dk_i \geq Dk_{DC_{cmp}} \tag{3.8}$$

$$DC_{i+2} \xrightarrow{anahtar} DC_i \tag{3.9}$$

Veri işleme süreci devam eden kümelerden, ilerleme durumu t'den küçük olursa (7), işini bitiren kümelere anahtar dağıtımında DC_i'lerin farklılığını dikkate almak için, DC_i'lerin birim zamandaki indirge görevlerindeki veri işleme performansları hesaplanır. Hesaplama için, daha önceki eşle/indirge işindeki indirge görevinin, giriş veri boyutu, giriş anahtar sayısı, çıktı veri boyutu, çıktı anahtar sayısı, işin türü, indirge fonksiyonunun çalıştığı düğümün RAM ve CPU değerleri ile indirge görevlerinin çalışma süreleri dikkate alınır.

İndirge görevlerinin girdi veri boyutuna girdi kayıt sayısına oranlanarak girdi ağırlığı (3.10), çıktı veri boyutunu çıktı kayıt sayısına (3.11) oranlanarak çıktı ağırlığı bulunur.

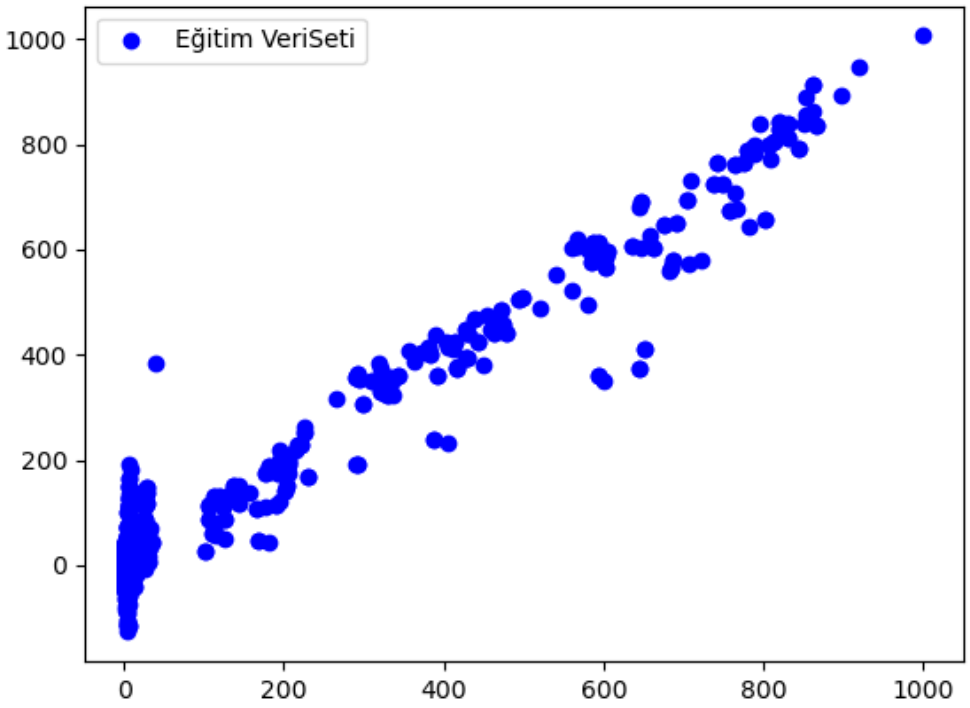
$$Wi_i = \frac{SRi_i}{CRi_i} \tag{3.30}$$

$$Wo_i = \frac{SRo_i}{CRo_i} \tag{3.11}$$

RAM ve CPU'nun hesaplama gücüne doğrudan etki ettiğinden, farklı RAM ve CPU

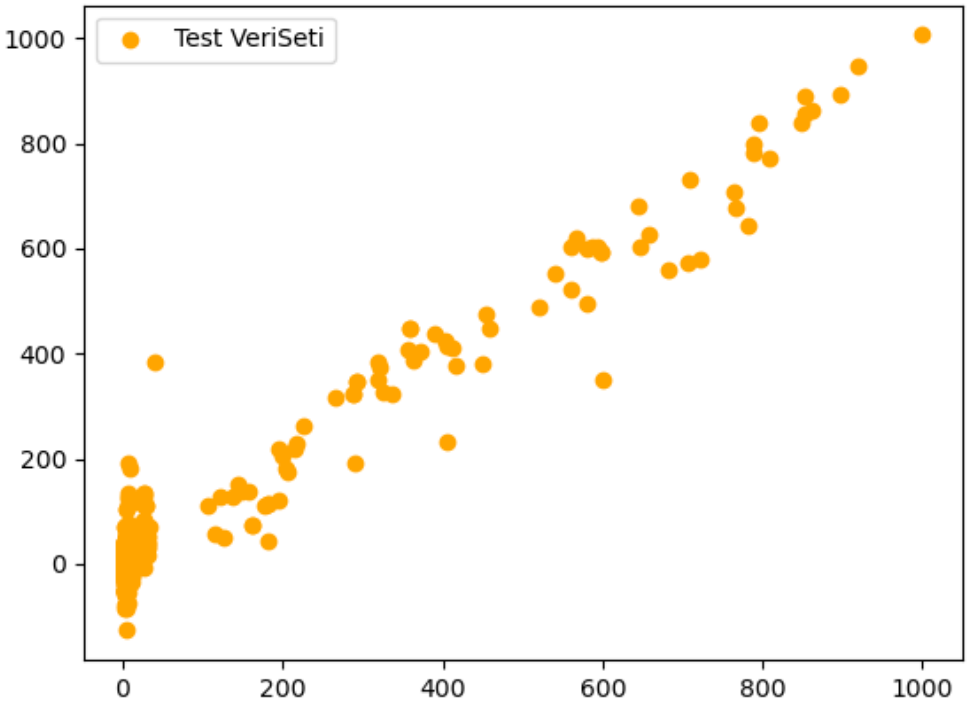
çekirdek sayısına sahip aynı veri boyutunda, farklı veriler ile, oluşturulan test ortamında 2596 test gerçekleştirilmiştir.

Denklemi oluşturan değişkenler arasındaki ilişkiyi gösteren katsayıların hesaplanması için altıncı dereceden (5 değişkenli) polinomal regresyon modeli kullanılmıştır. Test verilerinin %75'i ile sisteminin eğitimi gerçekleştirilmiş olup, verilerin %25'i ile sistemin tahmin performansı test edilmiştir. Eğitim verileri üzerindeki tahmin grafiği Şekil 3.3'te, test verileri üzerindeki tahmin grafiği Şekil 3.4'te ve gerçek ve tahmin verilerinin örtüşme grafiği ise Şekil 3.5'te verilmiştir.

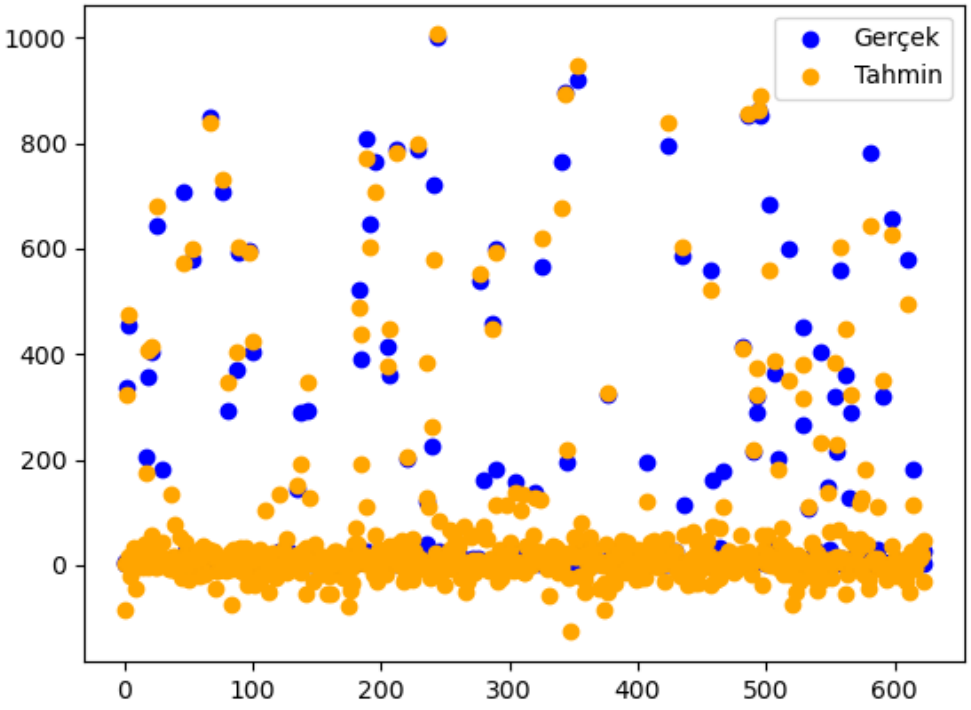


Şekil 3.3. Eğitim verilerinin tahminlerinin sonucu.

Polinomal regresyon makine öğrenme algoritmasının eğitim süresi 0,76 sn, tahmin süresi 1,4 ms olarak gerçekleşmiştir. Bu model sonucu 3002 adet katsayıya sahip bir denklem elde edilmiştir. Bu modelin performansı toplam hata kare skoru ile değerlendirilmiş olup, sonuç olarak %98 başarım elde edilmiştir.



Şekil 3.4. Test verilerinin tahminlerinin sonucu.



Şekil 3.5. Gerçek ve tahmin edilen verilerin örtüşme grafiği.

Polinomal regresyon ile katsayıları elde edilen denklem kullanarak, indirge görevlerinin çalıştırılacağı kümelerdeki düğümlerin maliyetleri hesaplanır.

Her bir anahtar/değer verisinin taşınacağı kümeleri belirlemek için, maliyetleri de hesaba katarak anahtar dağılımının belirlenmesi sonraki bölümde sunulmuştur.

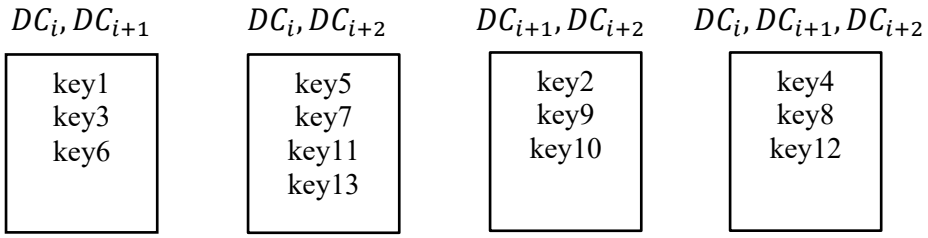
3.2. ANAHTAR DAĞITIMI

Bu kısımda kümeler arası anahtar/değerlerin karıştırma adımları sunulmuştur. Bölüm 3.1'de hesaplanan maliyete ve [13]' de sunulduğu gibi, anahtarın bulunduğu kümede olması koşuluyla anahtar dağıtımı kümeler arasında gerçekleştirilir.

Bölüm 3.1'de; kümeler eşle/indirge işini bitirdiğinde DCM'ye bildirim gönderir. DCM'de tamamlanan iş sayısı en az 2 olduğunda, kümelerdeki anahtarlara ait çıktıların boyutları DCM tarafından kontrol edilir. Anahtarlar, veri boyutu daha küçük olan kümeden daha büyük olan kümeye taşınır. Kümelerin indirge görevlerinin verilerine göre maliyetleri belirlenir.

Anahtar dağıtımı adımları şöyledir:

1. adım: Kümelerde toplanan ortak anahtarlar bir eşle/indirge işi ile belirlenir. Aynı anahtara sahip kümelerdeki anahtarlar gruplanır. Örneğin Şekil 3.6'da key1, key3 ve key6 hem DC_i hemde DC_{i+1} de bulunduğundan bir grup; key5, key7, key11 ve key13 hem DC_i hemde DC_{i+2}'de bulunduğundan bir grup; key2, key9, key10 key13 hem DC_{i+1} hemde DC_{i+2} var bulunduğundan bir grup ve key4, key8, key12, üç kümede (DC_i, DC_{i+1}, DC_{i+2}) de bulunduğundan bir grup oluşturmaktadır.



Şekil 3.6. Aynı anahtarlara sahip kümelerin grupları.

2. adım: Gruplama, kümelerdeki maliyetleri hesaplamayı sağlamaktadır. Her gruptaki veri işleme maliyetlerine göre kümelere atanacak anahtarların sayısı belirlenmektedir. Kümelerdeki ortak anahtarların toplandığı grup sayısı aşağıdaki formülle belirlenir. (3.12)

$$2^{D_{cmp}} - \left(D_{cmp} + 1\right) \tag{3.42}$$

D_{cmp} işini tamamlayan küme sayısını gösterdiğinden, formül ile $2^{D_{cmp}}$ tane farklı alt küme oluşturulur. $\left(D_{cmp} + 1\right)$ ile alt kümelerde sadece bir kümeye ait olan anahtar grubu ve boş küme çıkartılır. Bir anahtar sadece kendi veri merkezindeki kümede bulunuyorsa anahtarların gruplanmasına, sonrasında karıştırma aşamasında anahtar/değer çiftinin işlenmesine ve kümeler arasında taşınmasına gerek yoktur. Bu nedenle maliyet hesabına katılmaz.

3. adım: Birden fazla kümede bulunan anahtarların dağılımları, önceki bölümde hesaplanan maliyet katsayılarına göre gerçekleştirilir.

Tüm kümelerin, başlangıç maliyetleri 0'dır.

Her grubu oluşturan kümelerin işleyeceği anahtarları belirlemek için, grubu oluşturan kümelerin birbirlerine olan maliyetlerinin ters orantıları dikkate alınır. Böylelikle maliyetlerin yaklaşık olarak yakın olması amaçlanır. Her gruptaki anahtarlar, kümelerin maliyetlerine göre atanır. Bunlar, denklem 3.13 ve 3.14'te sunulmuştur:

$$\frac{Cost_i}{Cost_{i+1}} = \frac{Ck_{i+1}^{DC_i, DC_{i+1}}}{Ck_i^{DC_i, DC_{i+1}}} \tag{3.53}$$

$$Ck_i^{DC_i, DC_{i+1}} + Ck_{i+1}^{DC_i, DC_{i+1}} = Count(DC_i, DC_{i+1}) \tag{3.14}$$

4. adım: Her kümeye anahtarlar atandığında oluşan maliyetler hesaplanır. Öncelikle en küçük kümelerin oluşturduğu grupların anahtarlarının dağılımı gerçekleştirildikten sonra oluşan, kümelerin maliyetleri denklem 3.15'te verildiği gibi hesaplanır.

$$TCost_i = Ck_i^{DC_i, DC_{i+1}} * Cost_i \tag{3.65}$$

5. adım: Küçük küme gruplarından daha büyük gruplardaki kümelerin maliyetlerini eşitlemek için en büyük maliyete sahip küme ile diğer kümelerin maliyetinin farkı denklem 3.16'te gösterildiği gibi hesaplanır.

$$Diff_{i+1,i} = Max(TCost_{i+1}) - TCost_i \tag{3.76}$$

$0 \leq i \leq DC_{cmp}, \forall i \in I$

6. adım: Maliyeti düşük olan kümelere, en çok kümenin bulunduğu ortak anahtar grubundan, atanacak anahtar sayısı denklem 3.17'da gösterildiği gibi belirlenir.

$$DiffCount\left(Ck_j^{DC_i,DC_{i+1},DC_{i+2}}\right) = \frac{Diff_{i+1,j}}{Cost_j} \tag{3.87}$$

7. adım: Düşük maliyetli olan kümelere atanacak anahtar sayıları belirlendikten sonra, kalan anahtarlar maliyet oranlarına göre anahtar sayıları belirlenerek anahtar dağılımları Denklem 3.18'de gösterildiği gibi hesaplanır.

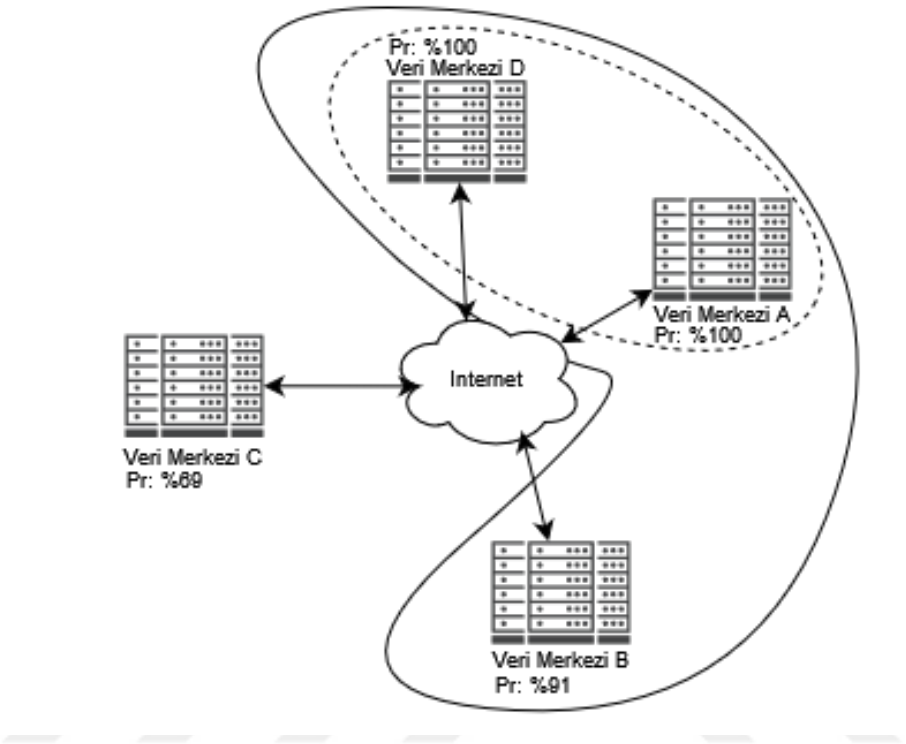
$$\begin{aligned} Ck_i^{DC_i,DC_{i+1},DC_{i+2}} + Ck_{i+1}^{DC_i,DC_{i+1},DC_{i+2}} + Ck_{i+2}^{DC_i,DC_{i+1},DC_{i+2}} \\ = Count(DC_i, DC_{i+1}, DC_{i+2}) \end{aligned} \tag{3.98}$$

Anahtar dağılımları kümelere gönderilir. Hedef kümeye, kendi kümesine ait anahtarlar gönderilmez. Dağıtılan anahtarlar, karıştırma için kümeler üzerinde anahtar/değer çiftleriyle eşleştirilir ve ait olduğu kümeye gönderilmek üzere karıştırma gerçekleştirilir. Karıştırma tamamlandıktan sonra veriler üzerinde indirge fonksiyonu çalıştırılır.

Yöntemimizi Şekil 3.7'de gösterilen örneği kullanarak açıklayalım. Her DC'de bir küme bulunmaktadır. Farklı coğrafi konumlarda bulunan dört küme (DC_A, DC_B, DC_C, DC_D) üzerinde bir eşle/indirge işi başlatalım.

DC_A ve DC_D üzerinde çalışan işler tamamlandığında DCM'ye bildirim gönderirler. DC_A'daki anahtar çıktı boyutu 110 br, DC_D'deki 150 br olduğunu varsayalım. DC_A'dan DC_D'ye anahtarlar gönderilir. DCM, devam eden kümelerin ilerleme durumlarını kontrol eder. Ön tanımlı bir değerden daha büyük ilerleme durumuna sahip bir DC varsa (%90 kabul ettiğimizi varsayalım); burada DC_B %91 olduğundan işi tamamlaması beklenir. Tamamladıktan sonra, DC_B'deki anahtar boyutunun 100 br olduğunu varsayalım. DC_D'deki anahtar boyutu 260 br olduğundan, DC_B'den DC_D'ye anahtarlar taşınır. İşine devam eden DC'ların iş durumları kontrol edilir. İlerleme durumu %90'dan büyük DC yok ise, işini tamamlayan DC'lerdeki kümelerin (DC_A, DC_B, DC_D) RAM, CPU, indirge

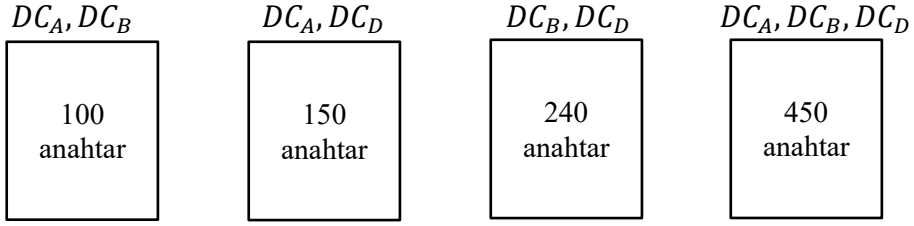
görevlerinin girdi ve çıktı ağırlıkları ve iş türü ile, önceden katsayıları hesaplanan denklemde, değişkenler yerlerine konularak maliyetler belirlenir. Kümelerin maliyetleri Çizelge 3.1'da verilmiştir.



Şekil 3.7. Önerilen yöntem için örnek topoloji.

Çizelge 3.1. DC'lerin maliyet tablosu.

Kümelerin Maliyetleri		
DC_A	DC_B	DC_D
1.5	3	2



Şekil 3.8. Ortak anahtarların gruplanması.

Kümeler arasında ortak olan anahtarlar belirlenir. Aynı anahtara sahip kümeler gruplanır (Şekil 3.8). En küçük gruplardan başlanarak maliyetlere göre anahtarlar dağıtılır.

Tablodaki maliyetlere göre DC_A, DC_B ve DC_D'nin maliyetleri sırasıyla 1.5, 3 ve 2 'dir. DC_A, DC_B bulunduğu grupta 100 adet anahtar vardır. DC_A'ya 66, DC_B'ye 34 anahtar atanır. DC_A ve DC_B'nin maliyetleri birbirine yakındır ve yaklaşık 100 olur. DC_A, DC_D'nin bulunduğu grupta 150 anahtar var. Bu grubu oluşturan maliyetlere göre DC_A'ya 85, DC_D'ye 65 anahtar atanır. DC_A ve DC_D'nin maliyetleri birbirine yakındır ve yaklaşık 130 olur. DC_B, DC_D'nin bulunduğu grupta 240 adet anahtar var. Bu maliyetlere göre DC_B'ya 96, DC_D'ye 144 anahtar atanır. DC_B ve DC_D'nin maliyetleri birbirine yakındır ve yaklaşık 288 olur. Bu 3 gruptaki anahtarların dağıtılması ile oluşan kümelerin maliyetleri toplamı $TCost_A$, $TCost_B$ ve $TCost_D$ sırasıyla, 230, 388 ve 418'dir. Tüm kümelerin ait olduğu grup hariç, diğer tüm anahtar gruplarının anahtar dağıtımı bu şekilde gerçekleşir. Son grupta ise kümelerin maliyetlerinin farkı ele alınarak dağıtım gerçekleşir. Önceki gruplarda dağıtılan anahtarlara göre $TCost_D$ en büyük maliyete sahiptir. Bu gruptaki anahtarların dağılımı belirlenmeden önce kümelerin maliyetleri eşitlenmelidir. DC_D'nın maliyeti DC_A'dan 188, DC_B'den 30 fazladır. DC_A 125 key ve DC_B'ye 10 anahtar atandığında maliyetler yaklaşık olarak eşitlenir. Bu grupta kalan 315 anahtar, maliyet kümelerin maliyetlerine göre dağıtılır. Son gruptaki kalan anahtarların dağılımı DC_A, DC_B ve DC_D'ye sırasıyla, 140, 70 ve 105 olur. Sonra anahtar dağılımları kümelere gönderilir. Kümelerde belirlenen anahtar/değerler çiftleri ait olduğu kümelere taşınarak karıştırma gerçekleştirilir. Daha sonra kümelerde indirge işi çalıştırılır. Bu grubu oluşturan kümelerin maliyetlerine göre anahtar dağıtımı sonrası anahtar sayıları Çizelge 3.2'de, maliyetleri de Çizelge 3.3'te sunulduğu gibi olur.

Çizelge 3.2. Anahtar dağıtımı sonrası DC'lerdeki anahtar sayıları.

Grup	DC	Anahtar Sayısı
Grup1	DC_A	66
	DC_B	34
Grup 2	DC_A	85
	DC_D	65
Grup 3	DC_B	96
	DC_D	144
Grup 4	DC_A	140+125
	DC_B	70+10
	DC_D	105

Çizelge 3.3'te gösterildiği gibi kümelerin maliyetleri yaklaşık eşitlenmiş olur.

Çizelge 3.3. Anahtarlar dağıtıldıktan sonra maliyetler

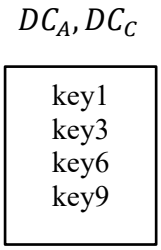
DC	Toplam Maliyet
DC_A	627.5
DC_B	630
DC_D	628

DC_A, DC_B ve DC_D veri merkezlerindeki kümelerde indirge işi başlatıldıktan sonra, işine devam tüm kümelerden bildirim beklenir. İlk faz tamamlandıktan sonra, DC_A ve DC_C üzerinde çalışan işler tamamlandığında DCM'ye bildirim gönderir. DC_A'daki anahtar çıktı boyutu 30 br, DC_C'deki 240 br olduğunu varsayalım. DC_A'dan DC_C'ye anahtarlar gönderilir. DCM, devam eden kümelerin ilerleme durumlarını kontrol eder. Ön tanımlı bir değerden daha yüksek ilerleme durumuna sahip bir DC varsa bunların kontrolü sağlanır. (%90 kabul ettiğimizi varsayalım), DC_B ve DC_D'nin iş ilerleme durumları sırasıyla %98 ve %99 olduğundan, işleri tamamlamaları beklenir. Tamamladıktan sonra, DC_B'deki anahtar boyutunun 60 br ve DC_D'deki anahtar boyutu 45 br olduğundan, DC_B'den DC_C'ye ve DC_D'den DC_C anahtarlar taşınır. Tüm kümeler işleri tamamladığından DC'lerin (DC_A, DC_B, DC_C, DC_D) RAM, CPU, iş türü, indirge görevinin girdi ve çıktı ağırlıklarıyla ağırlıkları oluşturulan denklem ile maliyetler hesaplanarak Çizelge 3.4'te sunulmuştur.

Çizelge 3.4. DC'lerdeki kümelerin maliyet tablosu (2.durum).

Kümelerin Maliyetleri			
DC_A	DC_B	DC_C	DC_D
1.5	3	1	2

Kümeler arasında ortak olan anahtarlar belirlenir. Aynı anahtara sahip kümeler olarak gruplanır (Şekil 3.9).



Şekil 3.9. Ortak anahtarların gruplanması.

Önceki adımda DC_A, DC_B ve DC_D arasında karıştırma ve indirge gerçekleştirildiğinden, bu kümeler arasındaki anahtarlar benzersizdir. Bu nedenle bu kümeler arasında grup oluşmaz. Ortak anahtarlar sadece DC_A ve DC_C arasında bulunduğundan tek grup oluşur. DC_A, DC_C'nin maliyetleri sırasıyla 1.5 ve 1'dir. Bu kümelerin bulunduğu grupta 50 adet anahtar vardır. Maliyetlere göre DC_A'ya 20, DC_C'ye 30 anahtar atanır. Sonra anahtar dağılımları DC_A ve DC_C'ye gönderilir. Kümelerdeki anahtar/değer çiftleri eşleştirildikten sonra, kümeler arasında karıştırma gerçekleşir. Bu grubu oluşturan kümelerin maliyetlerine göre anahtar dağılımları Çizelge 3.5'deki gibi gösterildiği gerçekleştirilmiş olur.

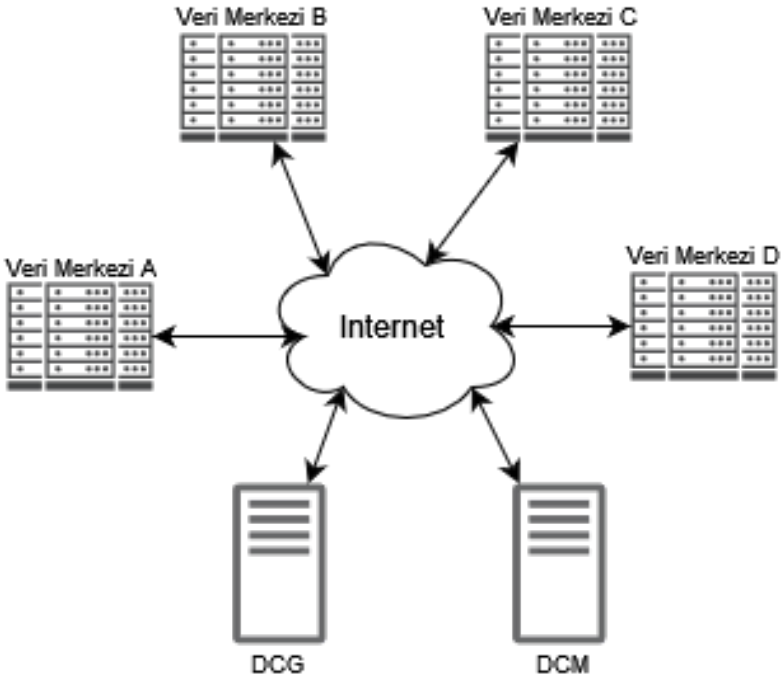
Çizelge 3.5. Anahtar dağıtımı sonrası DC'lerdeki kümelerde oluşan anahtar sayıları.

Grup	DC	Anahtar Sayısı
Grup1	DC_A	20
	DC_C	30

Karıştırma tamamlandıktan sonra indirge işi çalıştırılır. Tüm kümeler karıştırmayı gerçekleştirdiğinden kümelerdeki eşle/indirge işi tamamlanmış olur. Ayrıca ilk fazda DC_A, DC_B ve DC_D shuffling ve reduce işlerini tamamladığından, son fazda bu kümeler arasında karıştırma gerçekleşmez. Son aşamada hem kümeler arası karıştırma veri trafiği ve bunlara bağlı olarak işin tamamlanma süresi azaltılmış olur.

4. GSELF-MAPREDUCE YÖNTEMİNİN PERFORMANSININ DEĞERLENDİRİLMESİ

Önerilen GSelf-MapReduce yöntemi, dağıtık DC'lerden oluşan yapıda uygulandı. Bunun için 4 adet DC'den oluşan bir yapı oluşturuldu. Bu yapıda, her biri farklı donanıma ve veri boyutuna sahip kümelerden oluşan DC'ler vardır. Bu DC'ler, Düzce Üniversitesi bünyesindeki sunucularda, her biri farklı ağlar üzerinde kurulmuştur. Her bir sunucu düğüm bağımsız DC olarak kabul edilmiştir. DC'lerin yanı sıra DC'lerin iş akışlarının devamlılığının sağlanması için DCM bulunmaktadır. (Şekil 4.1)



Şekil 4.1 Test ortamı.

Yapıdaki DC'lerin her birinin donanım özellikleri, bant genişliği Çizelge 4.1'de, veri boyutları Çizelge 4.2'de verilmiştir.

Test uygulamaları Ubuntu 20.04 LTS işletim sisteminde çalışan Hadoop 3.2.4'te gerçekleştirilmiştir.

Çizelge 4.1 DC'lerin donanım özellikleri tablosu.

Veri Merkezleri	Donanımlar	Bant Genişliği (Mbit)
Veri Merkezi A (DC_A)	Intel Xeon E5-2680 2.50GHz 2 CPU 6 GB RAM	20
Veri Merkezi B (DC_B)	Intel Xeon E5-2670 2.60GHz 4 CPU 4 GB RAM	20
Veri Merkezi C (DC_C)	Intel Xeon Gold 5218R 2.10GHz 4 CPU 2 GB RAM	20
Veri Merkezi D (DC_D)	Intel(R) Xeon E5-26602.60GHz 2 CPU 4 GB RAM	20
DCM	Intel Xeon E5-2680 2.50GHz 2 CPU 2 GB RAM	-

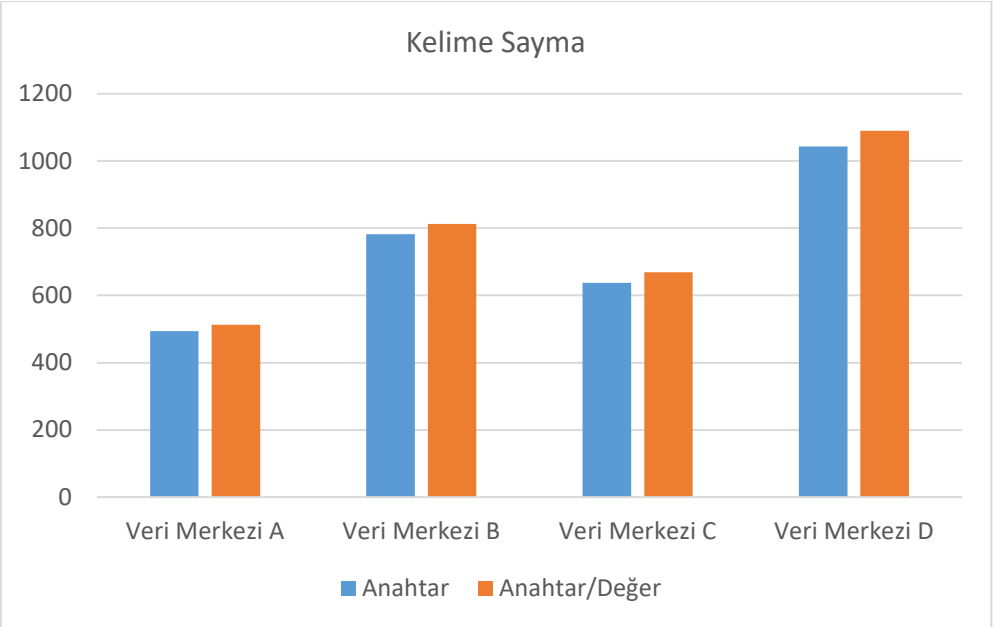
Çizelge 4.2 DC'lerdeki her bir uygulama için veri boyutu.

Veri Merkezleri	Kelime Sayma	Ters Dizin	Komşuluk Listesi
Veri Merkezi A	4G	4G	7.5G
Veri Merkezi B	6G	7.5G	6.5G
Veri Merkezi C	6.5G	5G	4.2G
Veri Merkezi D	9G	12G	8.3G

Akademik çalışma ortamlarındaki donanım veri, veri boyutu sınırlamaları nedeniyle donanım ve veri boyutları büyük seviyelerde ele alınamamaktadır.

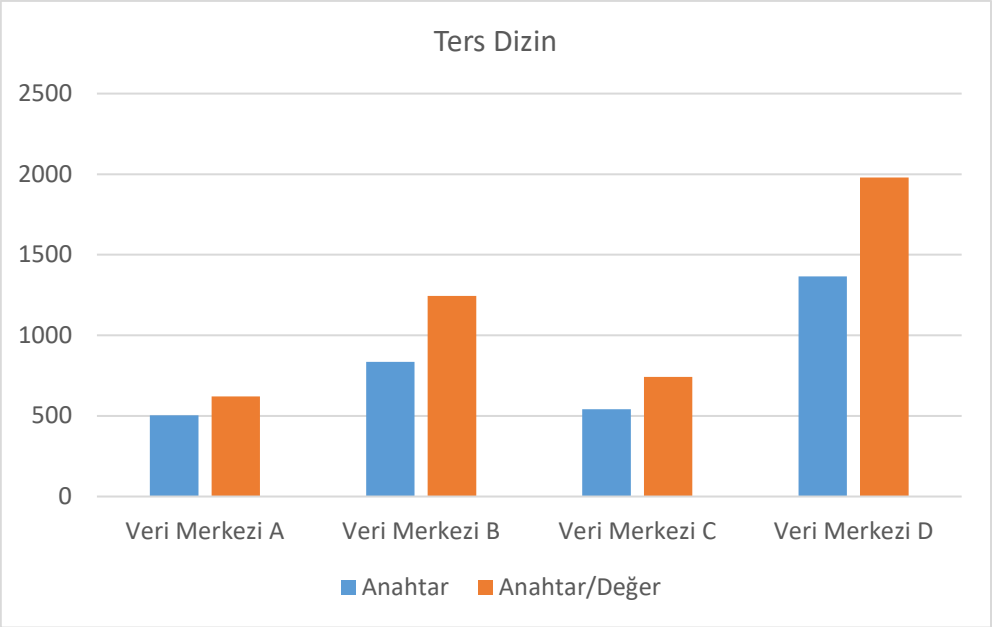
Veri seti olarak Puma [94], yöntemi test etmek için ise kelime sayma (word count), ters dizin (inverted index), ve komşuluk listesi (adjacency-list) uygulamaları kullanılmıştır.

Test ortamındaki 4 DC'de kelime sayma, ters dizin ve komşuluk listesi uygulamaları çalıştırılmıştır. Çalıştırıldıktan sonra her DC'de oluşan anahtar ve anahtar/değer boyutları; kelime sayma için Şekil 4.2'de, ters dizin için Şekil 4.3'de komşuluk listesi için Şekil 4.4'te verilmiştir.



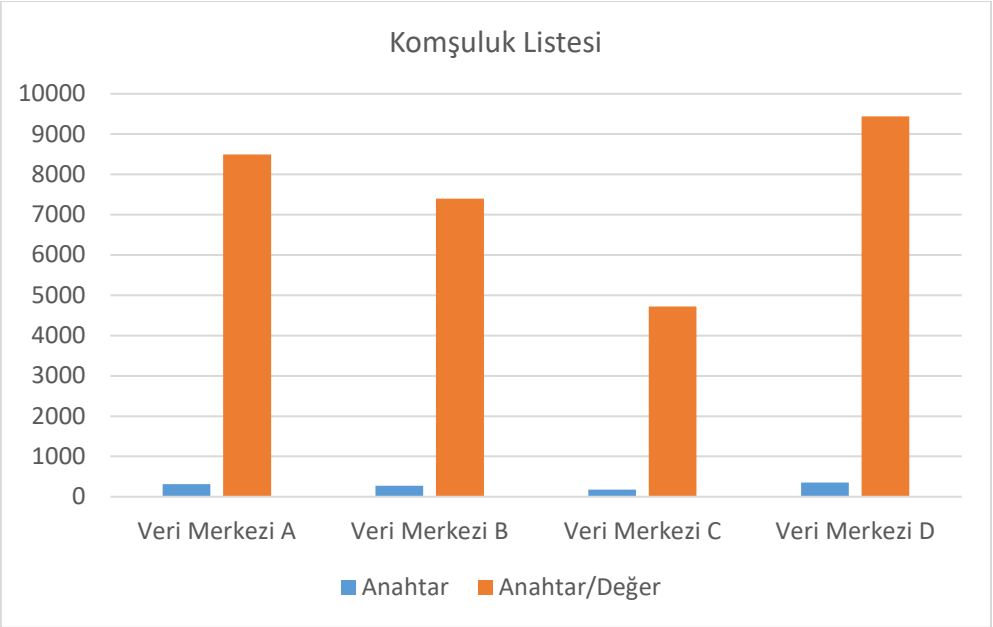
Şekil 4.2 Kelime sayma uygulamasının çıktı anahtar ve anahtar/değer boyutları (mb).

Şekil 4.2'de, kelime sayma uygulamasının çıktı anahtar ve anahtar/değer boyutlarının birbirine yakın olduğu görülmektedir.



Şekil 4.3 Ters dizin uygulamasının çıktı anahtar ve anahtar/değer boyutları (mb).

Şekil 4.3'de görüldüğü gibi ters dizin uygulamasının çıktı boyutlarında anahtar boyutu ve anahtar/değer boyutundan yaklaşık %25 oranında daha azdır.



Şekil 4.4 Komşuluk listesi uygulamasının çıktı anahtar ve anahtar/değer boyutları (mb).

Şekil 4.4'de, komşuluk listesi uygulamasının çıktı anahtar boyutu, anahtar/değer boyutundan oldukça küçüktür.

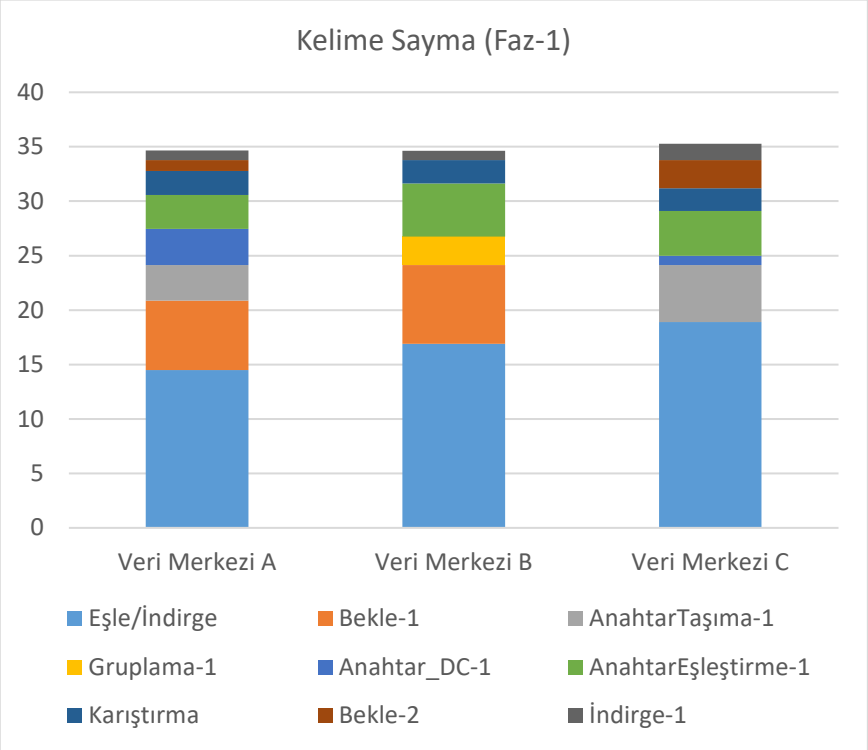
Cross-MapReduce ve Hiyerarşik Hadoop yaklaşımında olduğu gibi bizim önerdiğimiz yöntemde de öncelikle her DC'de eşle/indirge test uygulaması çalıştırılmıştır. Hiyerarşik Hadoop yaklaşımında her bir DC'de, eşle/indirge işi çalıştırılıp tamamlandıktan sonra, oluşan tüm veriler küresel indirgeyici olarak seçilen bir DC'ye gönderilerek indirge işi çalıştırılmıştır.

Cross-MapReduce'da ise sadece anahtarları içeren çıktılar ana DC'ye gönderilerek GRG oluşturulur. Bu GRG'ye göre belirlenen anahtarlar DC'lere gönderilir. Sonrasında, DC'lerdeki anahtar/değer çiftleri birden fazla küresel indirgeyiciye gönderilir.

GSelf-MapReduce'da ise, belirli bir eşik değerine göre işini tamamlayan DC'lerden, toplam anahtar boyutu büyük olan DC'ye anahtarlar gönderilir. DC'lerin indirge fonksiyonunun çalıştırma performansına göre anahtar dağılımları gerçekleştirilir. Sonrasında anahtar dağılımları DC'lere gönderilir. Bu dağılımlara göre DC'ler karıştırma işlemini gerçekleştirir. Karıştırma işlemi bittikten sonra DC'lerdeki veriler üzerinde indirge işi çalıştırılır (Faz-1). Sonrasında tüm DC'ler işini bitirene kadar, Faz-1'deki aşamalar gerçekleştirilir.

4.1. KELİME SAYMA UYGULAMASI İLE GSELF-MAPREDUCE YÖNTEMİNİN PERFORMANSININ DEĞERLENDİRİLMESİ

Kelime sayma işini ilk olarak DC_A, 14.5 dk'da tamamlar. DCM'ye bildirim gönderir. DCM başka bir DC'den bildirim bekler. Sonra DC_B, 16.9 dk'da işini bitirdiğini bildirir. DC'ler işi bitirdiğinde sadece anahtar ve anahtar/değer olarak iki çıktı üretir. DCM, DC_A ve DC_B arasındaki anahtarların boyutlarını karşılaştırır. Anahtar boyutu DC_A'da daha küçük olduğundan DC_A'dan DC_B'ye anahtarlar transfer edilir. DCM işine devam eden DC'lerin iş durumlarını kontrol eder. İş durumları %60'den fazla ise DC'lerin işini tamamlamasını bekler. DC_C'nin ilerleme durumu %94.7, DC_D'ninki %52.1 dur. DC_C işini tamamlar. DC_B'deki toplam anahtar boyutu ile DC_C'nin anahtar boyutu karşılaştırılır. DC_C'nin anahtar boyutu daha küçük olduğundan DC_C'den DC_B'ye anahtarlar taşınır. Sonra, DC_B'de; DC'lerdeki ortak anahtarlar gruplanır.



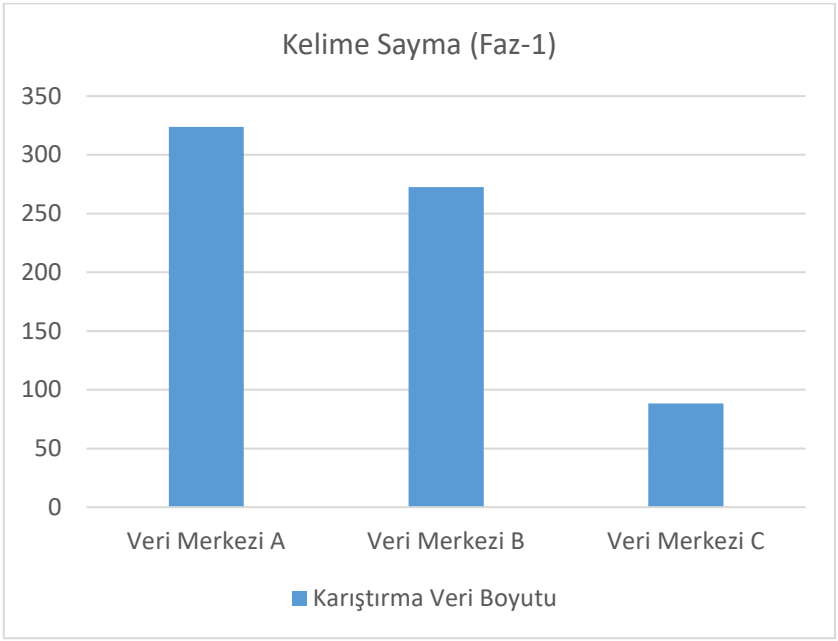
Şekil 4.5 Kelime sayma uygulaması faz-1 tamamlanma süresi (dk).

İşini bitiren üç DC'nin (DC_A, DC_B, DC_C) eşle/indirge işindeki indirge fonksiyonlarının giriş veri boyutu ve giriş anahtar/değer çifti sayısı, çıktı veri boyutu ve çıktı anahtar/değer çifti sayısı, RAM miktarı, CPU çekirdek sayısı ve iş türü alınıp, polinomal regresyon kullanılarak önceden katsayıları belirlenmiş denklem ile DC'lerin hesaplama kapasiteleri belirlenir. Gruplanan anahtarlar DC'lerin kapasitelerine göre dağıtılır. Anahtarlar DC'lere

taşınır. Her DC'de, diğer DC'lere gönderilecek anahtar/değerler belirlenir. DC'ler arasında karıştırma gerçekleştirilir. Karıştırma tamamlandıktan sonra indirge işi çalıştırılır.

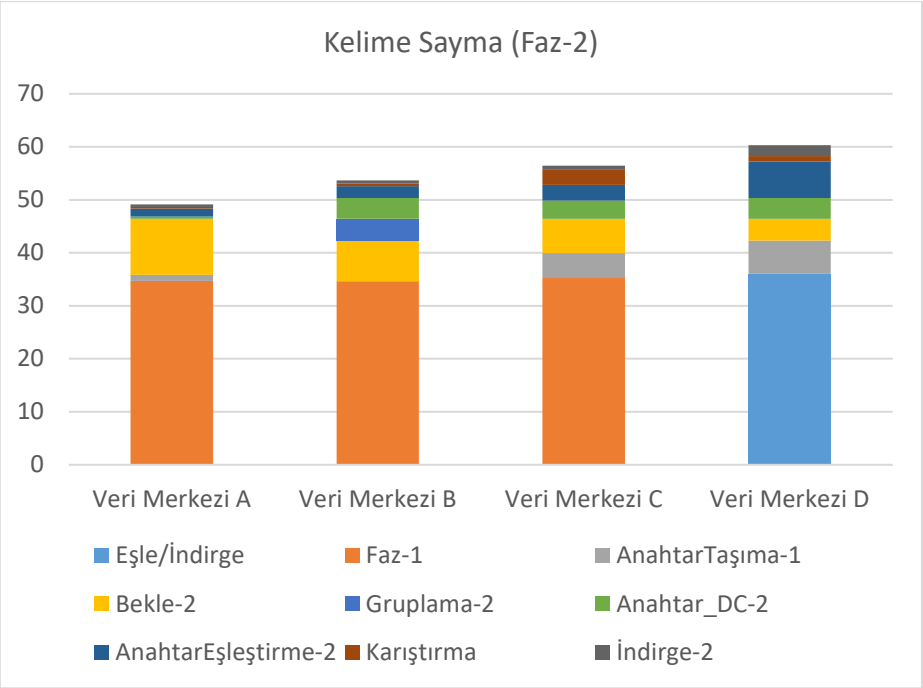
Daha sonrasında DC_B, Faz-1'i 34.64 dk'da tamamlar. DC_A 34.67 dk'da işini tamamlar. DC_A'nın anahtar boyutu DC_B'den daha küçük olduğundan, DC_A'dan DC_B'ye anahtarlar gönderilir. DC_C ve DC_D'nin ilerleme durumları kontrol edilir. İş ilerleme durumları sırasıyla; %99.1 ve %98.7 olduğundan işlerini tamamlamaları beklenir. Tüm DC'ler işi bitirdikten sonra Faz-1'deki aşamalar tekrar edilir. Tüm DC'ler aynı anda işlerini bitirdiğinden dağıtık eşle/indirge işi tamamlanır.

DC_A, DC_B ve DC_C'nin işleri tamamlama süreleri Şekil 4.5'te, veri merkezlerinden taşınan veri boyutu ile Şekil 4.6'da verilmiştir.



Şekil 4.6 Kelime sayma uygulamasında faz-1'de veri merkezleri arasında taşınan veri boyutları (mb).

Önerdiğimiz yöntemin, kelime sayma uygulaması için toplam çalışma süresi Şekil 4.7'te verilmiştir. Şekil 4.7'te görüldüğü gibi, DC_A, DC_B ve DC_C; veri merkezleri eşik değerine göre kendi aralarında ilk fazını; DC_D'nin başlangıçtaki eşle/indirge işinden daha önce tamamlayarak verilerini azaltmış olurlar.



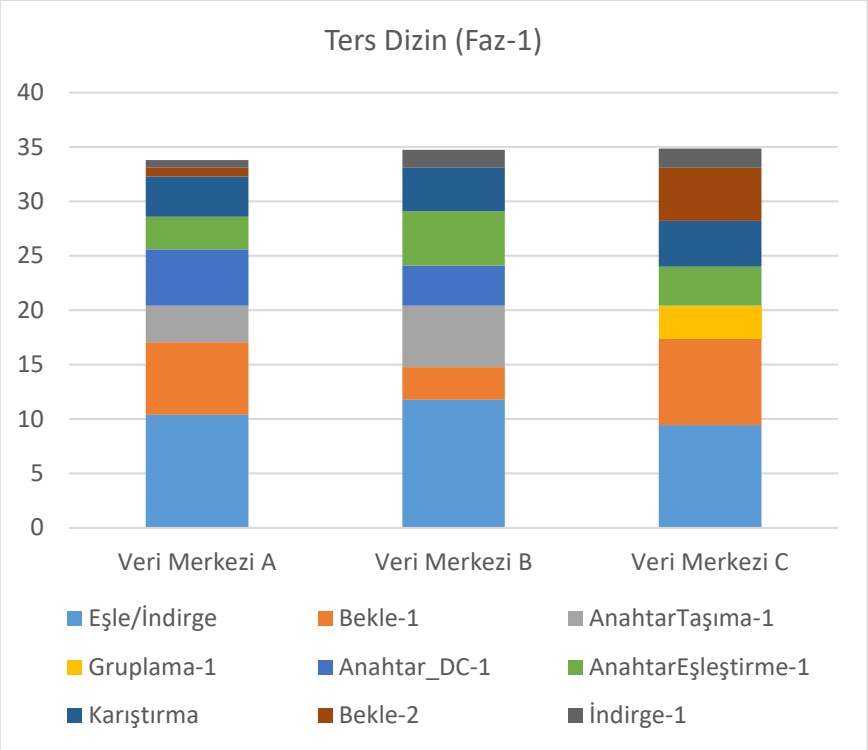
Şekil 4.7 Kelime sayma uygulamasının toplam çalışma süresi (dk).

4.2. TERS DİZİN UYGULAMASI İLE GSELF-MAPREDUCE YÖNTEMİNİN PERFORMANSININ DEĞERLENDİRİLMESİ

Ters dizin işi için ilk olarak DC_C, 9.45 dk'da tamamlar. DCM'ye bildirim gönderir. DCM başka bir DC'den bildirim bekler. Sonra DC_A, 10.38 dk'da işini bitirdiğini bildirir. DC'ler eşle/indirge işini bitirdiğinde anahtar ve anahtar/değer olarak iki çıktı üretir. DCM, DC_A ve DC_C arasındaki anahtarların boyutlarını karşılaştırır. Anahtar boyutu DC_A'da daha küçük olduğundan, DC_A'den DC_C'ye anahtarlar gönderilir. DCM işine devam eden DC'lerin iş durumlarını kontrol eder. İş durumları %60'den fazla ise veri merkezlerinin işini tamamlamasını bekler. DC_B'nin ilerleme durumu %97.7, DC_D'ninki %46.3'tür. DC_B işini tamamlar. DC_B'deki anahtar boyutu ile DC_C'nin toplam anahtar boyutu karşılaştırılır. DC_B'nin daha küçük olduğundan, DC_B'den DC_C'ye anahtarlar taşınır. Sonra, DC_C'de DC'lerden gönderilen ortak anahtarlar gruplanır. İşini bitiren üç DC'nin (DC_A, DC_B, DC_C) eşle/indirge işindeki indirge fonksiyonlarının giriş veri boyutu ve giriş anahtar/değer çifti sayısı, çıktı veri boyutu ve çıktı anahtar/değer çifti sayısı RAM miktarı, CPU çekirdek sayısı ve iş türü alınarak, polinomal regresyon kullanılarak önceden katsayıları belirlenmiş denklem ile DC'lerin maliyetleri hesaplanır. Gruplanan anahtarlar DC'lerin maliyetlerine göre dağıtılır. Anahtarlar DC'lere taşınır. Her DC'de diğer

DC'lere gönderilecek anahtar/değer çiftleri belirlenir. DC'ler arasında karıştırma gerçekleştirilir. İndirge fonksiyonu çalıştırılır.

DC_A, DC_B ve DC_C'nin çalışma süresi Şekil 4.8'te, veri merkezlerinden taşınan veri boyutu Şekil 4.9'da verilmiştir.

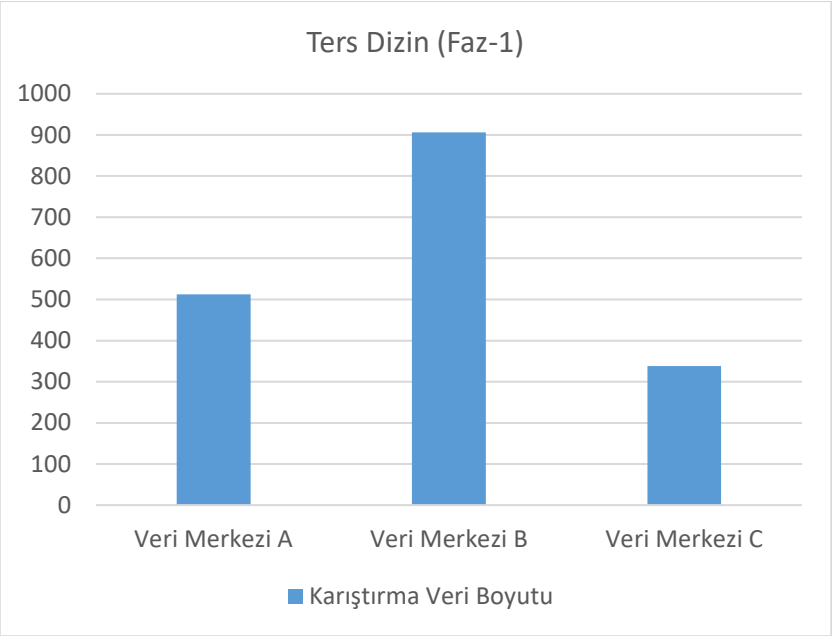


Şekil 4.8. Ters dizin uygulaması faz-1 çalışma süresi (dk).

Sonrasında DC_D, Faz-1'i 33.23 dk'da tamamlar. Sonra, DC_A işini 33.81 dk'da tamamlar. DC_A'nın anahtar boyutu DC_D'den daha küçük olduğundan, DC_A'dan DC_D'ye anahtarlar gönderilir. DC_B ve DC_C'nin ilerleme durumları kontrol edilir. İş ilerleme durumları sırasıyla; %99.3 ve %99.6 olduğundan işlerini tamamlamaları beklenir. Tüm DC'ler işi bitirdikten sonra, Faz-1'deki aşamalar tekrar edilir. Tüm DC'ler aynı anda işlerini bitirdiğinden dağıtık eşle/indirge işi tamamlanır.

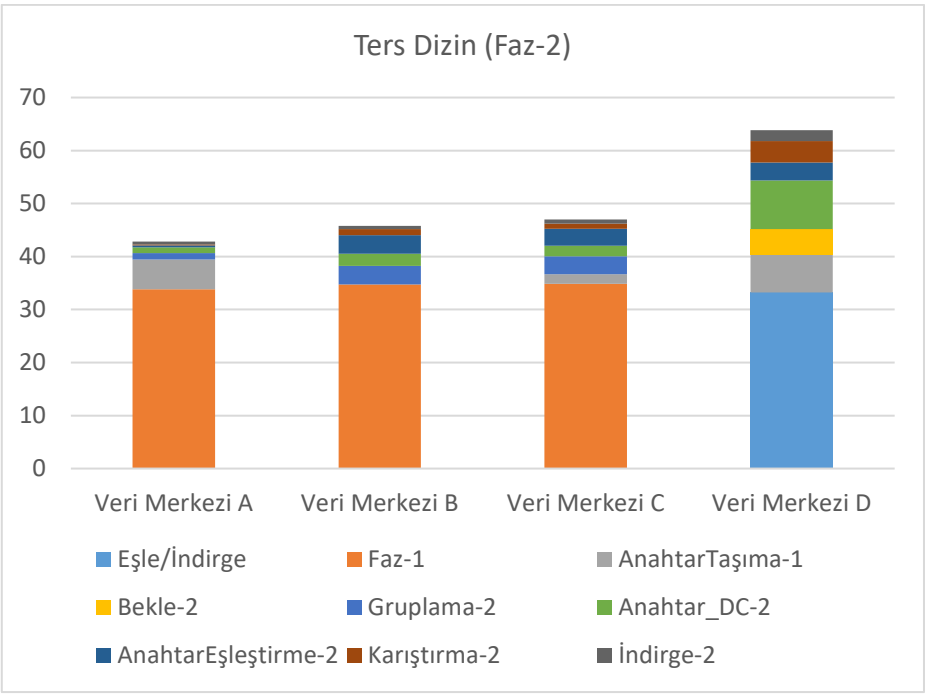
Önerdiğimiz yöntemin, ters dizin uygulaması için toplam çalışma süresi Şekil 4.10'da verilmiştir.

Şekil 4.10' da görüldüğü gibi, DC_D'nin iş ilerleme durumu ön tanımlı eşik değerinin altında olduğundan, DC_A, DC_B ve DC_C, karıştırma ve indirge aşamalarını DC_D'yi beklemeden gerçekleştirmişlerdir.



Şekil 4.9. Ters dizin uygulamasında faz-1'de veri merkezleri arasında taşınan veri boyutları (mb).

Böylelikle kendi aralarında ilk fazını DC_D'nin başlangıçtaki işinden daha önce tamamlayarak verilerini azaltmış olurlar.

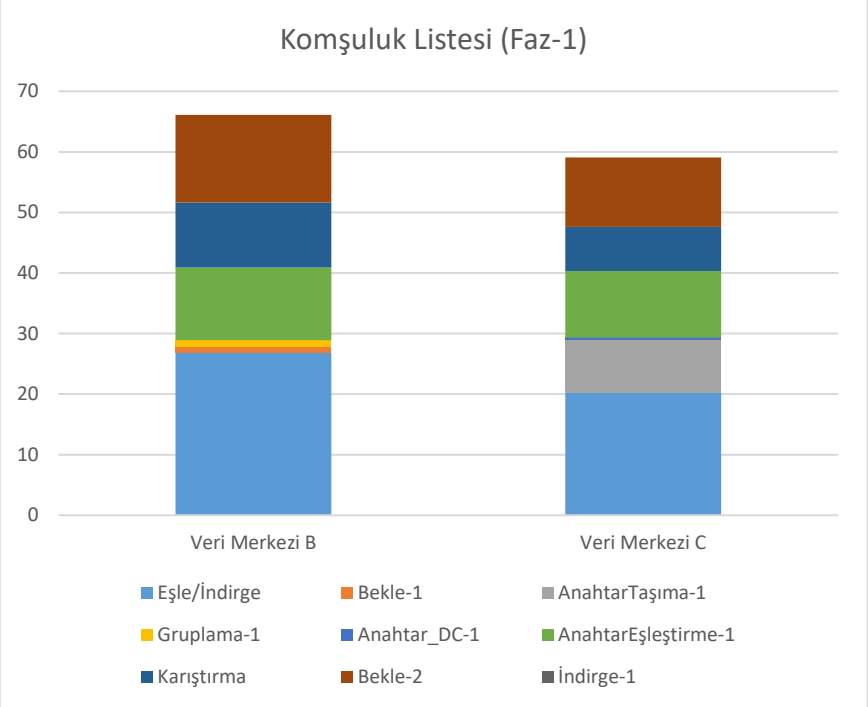


Şekil 4.10. Ters dizin uygulamasının toplam çalışma süresi (dk).

4.3. KOMŞULUK LİSTESİ UYGULAMASI İLE GSELF-MAPREDUCE YÖNTEMİNİN PERFORMANSININ DEĞERLENDİRİLMESİ

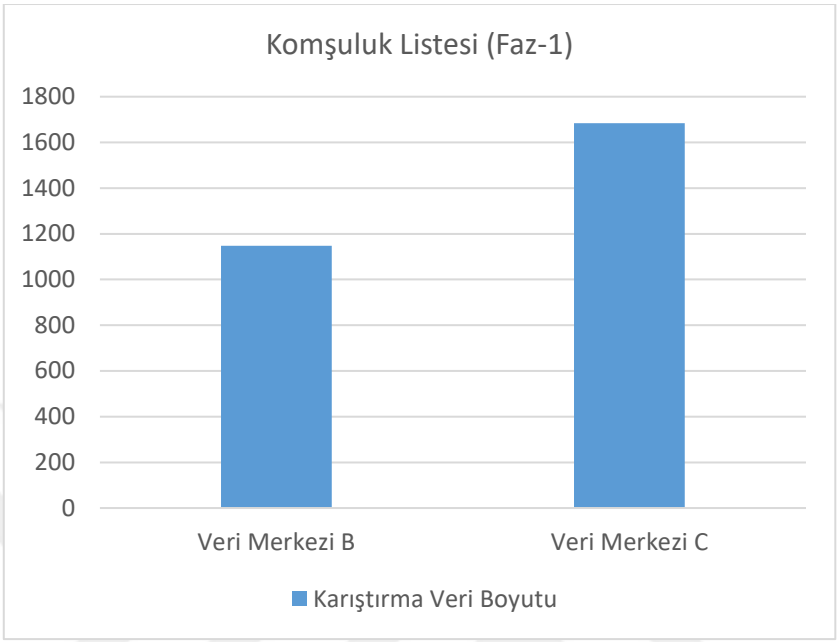
Komşuluk listesi işini ilk olarak DC_C, 20.17 dk'da tamamlar. DCM'ye bildirim gönderir. DCM başka bir DC'den bildirim bekler. Sonra DC_B, 26.77 dk'de işini bitirdiğini bildirir. DC'ler eşle/indirge işini bitirdiğinde sadece anahtar ve anahtar/değer olarak iki çıktı üretir. DCM, DC_B ve DC_C arasındaki anahtarların boyutlarını karşılaştırır. Anahtar boyutu DC_C'de daha küçük olduğundan, DC_C'den DC_B'ye anahtarlar gönderilir. DCM, işine devam eden DC'lerin iş ilerleme durumlarını kontrol eder. İş ilerleme durumları %60'den fazla ise DC'lerin işini tamamlamasını bekler. DC_A'nın ilerleme durumu %55.6 DC_D'ninki %48.4 dur. DC_B de, DC'lerden gönderilen ortak anahtarlar gruplanır. İşini bitiren iki DC'nin (DC_B, DC_C) indirge fonksiyonlarının giriş veri boyutu ve giriş anahtar/değer çifti sayısı, çıktı veri boyutu ve çıktı anahtar/değer çifti sayısı, RAM miktarı, CPU çekirdek sayısı ve iş türü alınarak polinomal regresyon kullanılarak önceden katsayıları belirlenmiş denklem ile DC'lerin hesaplama kapasiteleri hesaplanır.

Gruplanan anahtarlar DC'lerin hesaplama kapasitelerine göre dağıtılır. Anahtarlar DC'lere taşınır. Her DC'de diğer DC'lere gönderilecek anahtar/değer çiftleri belirlenir. DC'ler anahtar/değer çiftlerinin karıştırmasını gerçekleştirirler. Sonrasında DC_B ve DC_C'de, indirge işi çalıştırılır.

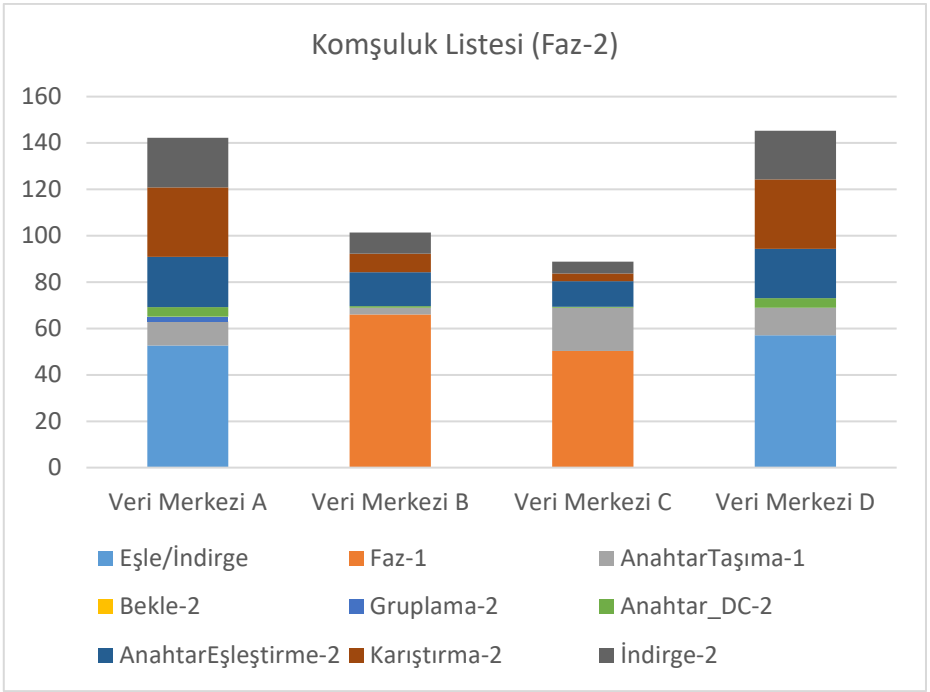


Şekil 4.11. Komşuluk listesi uygulaması faz-1 çalışma süresi (dk).

DC_A ve DC_D, eşik değerinin altında olduğundan, Şekil 4.11'da görüldüğü gibi DC_B ve DC_C karıştırma ve indirge işlerini DC_A ve DC_D'yi beklemeden gerçekleştirirler.



Şekil 4.12. Komşuluk listesi uygulamasında Faz-1'de veri merkezleri arasında taşınan veri boyutları (mb).



Şekil 4.13 Komşuluk listesi uygulamasının toplam süresi (dk).

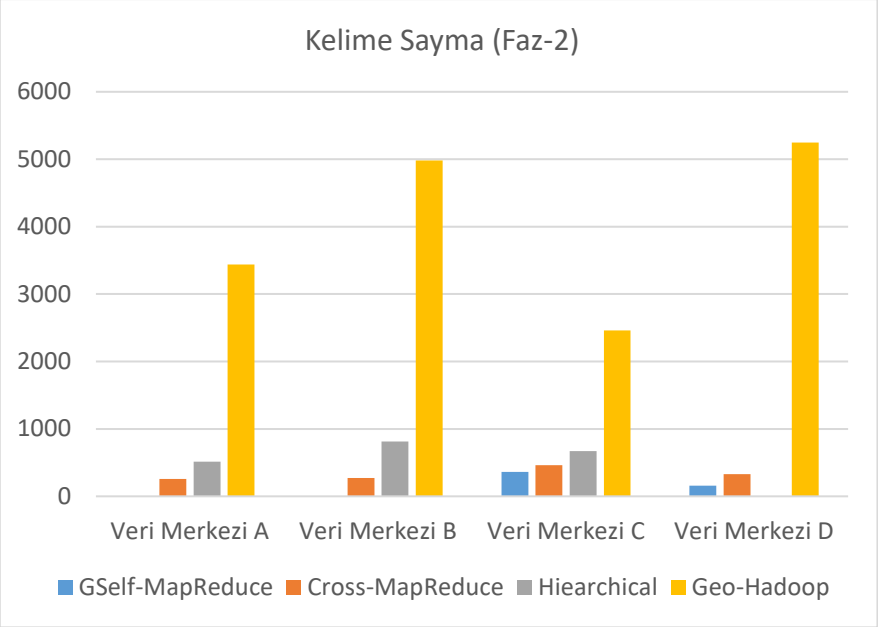
Sonra DC_C Faz-1'i 50.36 dk'da tamamlar. Ardından, DC_A 52.68 dk'da tamamlar. DC_C'nın anahtar boyutu DC_A'dan daha küçük olduğundan, DC_C'den DC_A'ya anahtarlar gönderilir. DC_B ve DC_D'nin iş ilerleme durumları kontrol edilir. İş ilerleme durumları sırasıyla; %94 ve %85 olduğundan işlerini tamamlamaları beklenir. İşini bitiren DC_D'nin anahtar boyutu küçük olduğundan, DC_D'den DC_A'ya, DC_B işini bitirdiğinde de DC_B'nin anahtar boyutu küçük olduğundan DC_B'den DC_A'ya anahtarlar taşınır. Tüm DC'lerin veri işleme kapasiteleri hesaplanır.

Anahtar dağılımları DC_A'da gerçekleşir. Anahtarlar DC'lere taşınır. Her DC'de diğer DC'lere gönderilecek anahtar/değer dağılımlarını belirlenir. DC'ler arasında karıştırma gerçekleştirilir. Tüm DC'lerde, indirge fonksiyonu çalıştırılır. Tüm kümelerde eşle/indirge işi 145.3 dk'da tamamlanır. Tüm DC'ler aynı anda karıştırma yaptığından dağıtık eşle/indirge işi tamamlanır.

4.4. GSELF-MAPREDUCE YÖNTEMİNİN DİĞER YÖNTEMLERLE KARŞILAŞTIRILMASINDAN ELDE EDİLEN SONUÇLAR

Cross-MapReduce, Hiyerarşik ve Geo-Hadoop yaklaşımlarının veri transferi tüm DC'ler işini bitirdiğinde gerçekleşir. GSelf-MapReduce birden çok fazdan oluşabildiğinden, son fazdaki karıştırmada gerçekleşen veri boyutu ile diğer yaklaşımların veri boyutlarını karşılaştırıyoruz.

Kelime sayma uygulamasında GSelf-MapReduce için, DC_A, DC_B ve DC_C veri merkezleri ilk fazda karıştırma ve indirge işini gerçekleştirdiklerinden bu veri merkezlerindeki anahtar/değer çiftleri azaltılmıştır. DC_A ve DC_B veri merkezlerinde bulunan anahtar DC_D'de bulunmadığından, DC_A ve DC_B'den, DC_D'ye herhangi bir karıştırma gerçekleşmeyecektir. Ortak anahtarlar DC_C ve DC_D arasında bulunduğundan, bu iki DC arasında karıştırma gerçekleşecektir. Şekil 4.14'te görüldüğü gibi karıştırma veri trafiğinin hacmi diğer yöntemlere göre daha azdır. Hiyerarşik yaklaşımda, bu çalışma için anahtar/değer boyutunun fazla olduğu DC_D küresel indirgeyici olarak seçilir. Böylelikle karıştırmada DC_D'deki veri boyutu dikkate alınmaz. Buna rağmen Cross-MapReduce, anahtarın bulunduğu DC'ler arasında karıştırma yaptığından veri trafiğinin hacmi Hiyerarşik ve Geo-Hadoop'a göre daha az olur.

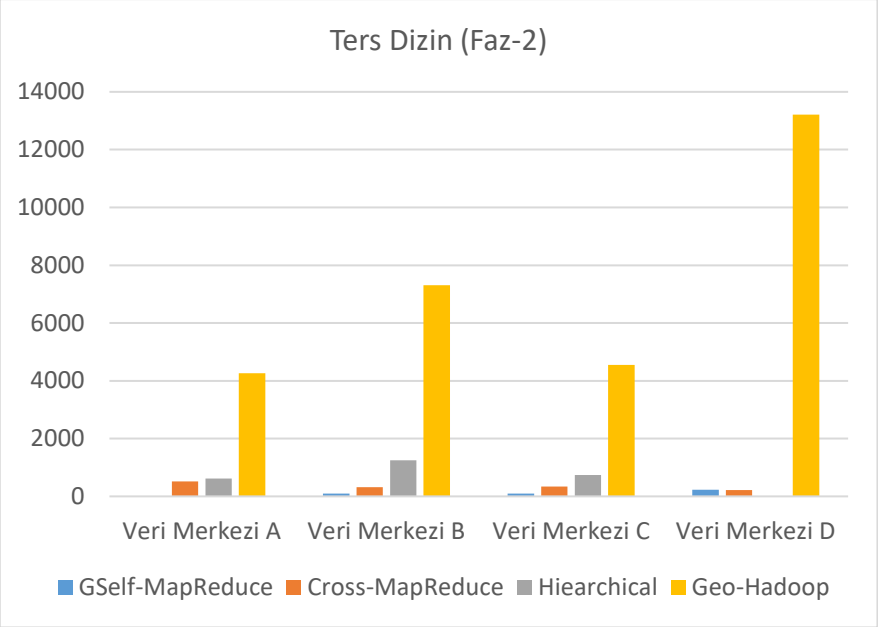


Şekil 4.14. Kelime sayma uygulaması için tüm veri merkezlerinden taşınan veri boyutu (mb).

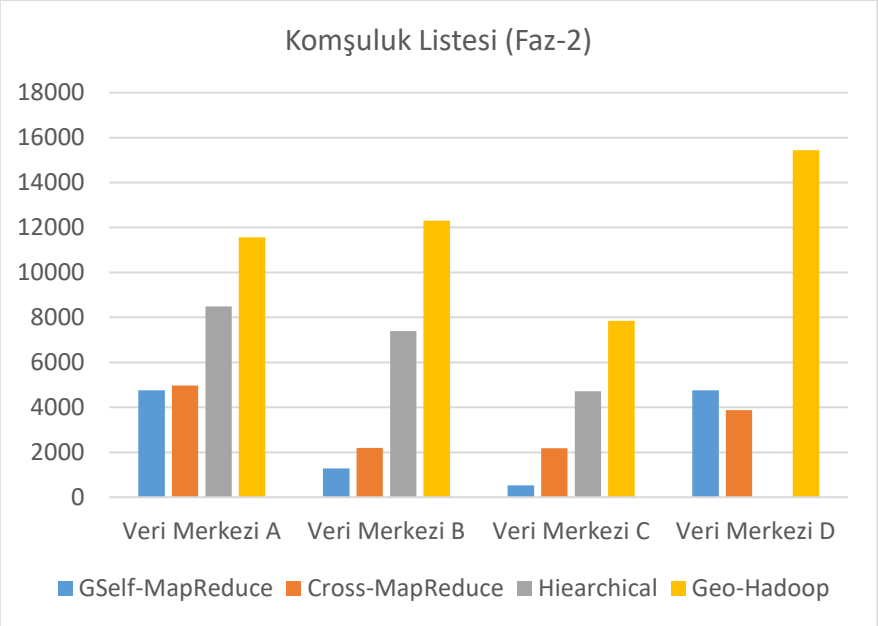
Ters dizin örneği için ilk fazda DC_A, DC_B ve DC_C, karıştırma ve indirgeme işlemini gerçekleştirdiklerinden bu veri merkezlerindeki anahtar/değer çiftleri azaltılmıştır. Son fazda DC_A'da bulunan anahtarlar DC_D'de bulunmadığından, DC_A ve DC_D arasında herhangi bir karıştırma gerçekleşmez. Ortak anahtarlar DC_B, DC_C ve DC_D arasında bulunduğundan, bu veri merkezleri arasında karıştırma gerçekleşir. DC_B ve DC_C arasındaki anahtar/değerler benzersiz olduğundan, karıştırma DC_B ve DC_D ile DC_C ve DC_D arasında gerçekleşir. Şekil 4.15'te görüldüğü gibi; GSelf-MapReduce'da, karıştırma veri trafiği diğer yöntemlere göre daha azdır. Bu uygulamada, Hiyerarşik yaklaşımda DC_D'deki anahtar/değer veri boyutu, diğer DC'lere göre büyük olduğu için DC_D, küresel indirgeyici olarak seçilir. Böylelikle karıştırma için DC_D'deki anahtar/değer çiftlerinin boyutu dikkate alınmaz. Hiyerarşik ile Cross-MapReduce yaklaşımlarının veri merkezlerindeki karıştırma veri trafiği hacmi neredeyse aynıdır. Geo-Hadoop ise diğer yaklaşımlara göre çok büyük veri trafiği hacmi oluşturur.

Komşuluk listesi örneği için ilk fazda DC_B ve DC_C karıştırma ve indirge işlemini gerçekleştirdiklerinden bu veri merkezlerindeki veriler azaltılmıştır. DC_B ve DC_C arasındaki anahtar benzersiz olmasına rağmen, tüm DC'ler arasında ortak anahtarlar bulunduğundan karıştırma gerçekleşecektir. Şekil 4.16 görüldüğü gibi, GSelf-MapReduce'da, DC_D'deki karıştırmada oluşan veri trafiği hacmi Cross-MapReduce'dan fazla olmasına rağmen, toplamdaki karıştırma veri trafiği hacmi Cross-MapReduce'dan

azdır. Hiyerarşik yaklaşım için DC_D'deki anahtar/değer boyutu fazla olduğundan, küresel indirgeyici olarak seçilir. Böylelikle karıştırmada DC_D'deki veri hacmi dikkate alınmaz. Hiyerarşik yaklaşımın, Cross-MapReduce'dan veri trafiği daha fazladır. Geo-Hadoop ise veri hacmi diğerlerine göre en fazla olan yaklaşımdır.



Şekil 4.15. Ters dizin uygulaması için tüm veri merkezlerinden taşınan veri boyutu (mb).



Şekil 4.16. Komşuluk listesi uygulaması için tüm veri merkezlerinden taşınan veri boyutu (mb)

Çizelge 4.3'de çalıştırdığımız uygulamaların toplam karıştırma veri trafiği hacmi ve eşle/indirge işlerinin tamamlanma süresini görülmektedir. Kelime sayma uygulamasındaki veri işleme süresi GSelf-MapReduce ve Hiyerarşik yaklaşımla, yaklaşık olarak aynıdır. Bunun en önemli nedeni çıktı anahtar boyutunun, çıktı anahtar/değer boyutuna yakın olmasındandır. Çünkü anahtarları bir DC'de gruplamak ve dağılımını sağlamak ve anahtar/değer çiftlerinin DC'lere taşımak zamansal maliyet ortaya çıkarır. Kelime sayma uygulaması için, Cross-MapReduce yaklaşımı Hiyerarşik yaklaşımdan daha az karıştırma veri trafiği hacmi çıkarmasına rağmen işin tamamlanma süresi daha fazladır. Ayrıca GSelf-MapReduce yönteminin karıştırma veri trafiği hacmi, en yakın yöntem olan Cross-MapReduce'dan %79 daha azdır. Çünkü ilk fazda toplam verinin %66'sı işlenmiştir.

Çizelge 4.3 Yöntemlerde çalıştırılan her bir uygulama için üretim süresi (dk) ve karıştırma veri boyutu (mb).

	Kelime Sayma		Ters Dizin		Komşuluk Listesi	
	Üretim Süresi	Karıştırma Veri Boyutu	Üretim Süresi	Karıştırma Veri Boyutu	Üretim Süresi	Karıştırma Veri Boyutu
GSelf-MapReduce	60.33	519	63.86	420	145.3	11323
Cross-MapReduce	77.2	1834	74.34	1384	182.31	13211
HMR	61.7	1994	82.56	2607	546.65	20610
Geo-Hadoop	202.1	16125	248	29327	2304	47152

Ters dizin uygulamasındaki GSelf-MapReduce'un veri işleme süresi Cross-MapReduce'dan daha kısadır. Bunun yanından karıştırma veri trafiği yine aynı yöntemden %69 daha azdır. Cross-MapReduce yaklaşımını Hiyerarşik ve Geo-Hadoop yaklaşımları izlemektedir. Geo-Hadoop'un eşle fazının çıktı veri boyutu, girdi veri boyutundan oldukça fazladır. Bu da yoğun ağ trafiğine kullanımına neden olduğundan, veri işleme süresini de oldukça uzamaktadır. Ayrıca GSelf-MapReduce'un ilk fazında toplam verinin %68.9'unu işlenerek azaltılmıştır.

GSelf-MapReduce yönteminin, komşuluk listesi uygulamasındaki karıştırma veri boyutuna, en yakın yaklaşım Cross-MapReduce'da görülmektedir. Son fazdaki karıştırma veri trafiği hacminde, Cross-MapReduce'dan yaklaşık %15 oranında daha az olduğu

görülmektedir. Komşuluk listesi uygulamasının indirge fazı, yoğun veri işleme gerektirir. Karıştırma veri boyutu, indirgeyicilerin işlediği veri boyutunu doğrudan etkilediğinden, verilerin işlenme süresi artar. Bunun en iyi örneğini Hiyerarşik yaklaşımda görülmektedir. Çünkü DC'lerden taşınan veriler tek DC'ye taşınıp küresel indirgeyici tarafından işlenir. Geo-Hadoop'ta ise veri merkezlerinde eşle aşamasının çıktısı karıştırma fazına katılacağından, veri merkezlerinden çıkan veri hacmi çok büyük olacaktır. Ayrıca GSelf-MapReduce yönteminde, ilk fazda toplam verinin %44.6'sı işlenerek azaltılmıştır.

5. SONUÇLAR

Dağıtık veri merkezlerinde verilerin depolanması ve işlemesi çağın gerekliliklerindendir. Çünkü verilerin üretildiği konumda depolanarak veri taşıma maliyetinin azaltılması önemlidir. Dağıtık verilerin işlenerek bir sonuç elde etmek istenebilir. Dağıtık verilerin işlenmesi için iki temel yaklaşım olan Geo-Hadoop yaklaşımı, Hiyerarşik yaklaşımın yanısıra literatürde bu yaklaşımlardan üretilen pek çok çalışma bulunmaktadır. Bu çalışmalardan iki temel yaklaşımın birlikte ele alınarak Cross-MapReduce adında hibrit bir yaklaşım sunulmuştur. Bu yaklaşımda dağıtık veri merkezlerinde, Hiyerarşik yaklaşımda olduğu gibi eşle/indirge işi çalıştırılır. İş tamamlandıktan sonra anahtarlar tek bir veri merkezinde toplanarak anahtar veri merkezinde olması koşuluyla anahtarlar dağıtılır. Anahtarlar dağıtıldıktan sonra DC'lere gönderilerek anahtar/değer çiftlerinin DC'ler arasında karıştırılması sağlanır. DC'lerde indirge işi çalıştırılır.

Bu çalışmada Cross-MapReduce değiştirilmiştir. Cross-MapReduce'daki anahtarların dağıtılması işi için veri merkezlerinin farklılığını ele alarak yeni bir yöntem sunuldu. Bu yöntemle veri merkezlerinin iş ilerleme durumlarını inceleyerek veri merkezlerinin kendi aralarında karıştırma yapmasını sağlandı. Böylelikle tüm veri merkezlerinin veri işlemesini bitirmesi beklenmeden, çıktı verileri azaltılarak karıştırma veri hacmi de azaltıldı. Ayrıca yoğun indirge işi çalıştırması gerektiren işler için işlenecek veri de azaltılmış oldu. Anahtarların dağıtılması için ortak anahtarlar gruplandı. Veri işleme maliyetlerine göre hızlı bir şekilde anahtarlar dağıtıldı.

GSelf-MapReduce'un, Cross-MapReduce'dan süre anlamında avantajlarından biri GRG oluşturmak için zaman harcanmamasıdır. Bunun yerine DC'lerdeki kümelerin veri işleme maliyetlerine göre anahtar sayılarına dayalı veri işleme yapması sağlandı.

GSelf-MapReduce yöntemi, Geo-Hadoop, Hiyerarşik Hadoop ve Cross-MapReduce ile karşılaştırıldı. Karşılaştırma için kelime sayma, ters dizin ve komşuluk listesi uygulamaları kullanıldı.

Bu çalışmada görüldüğü gibi, farklı donanım özellikleri ve veri boyutuna sahip DC'lerin veriyi işleme süreleri farklılık göstermektedir. Dağıtık eşle/indirge veri işleme yaklaşımlarından Cross-MapReduce Hiyerarşik Hadoop, Geo-Hadoop, tüm DC'lerin

eşle/indirge işini bitirmesini beklerken GSelf-MapReduce'da, ilerleme durumu belirli bir eşik değerinden küçük DC'lerin, kendi aralarında karıştırma ve ardından indirge yaparak verilerin azaltılması sağlanır. Böylelikle veri boyutu diğerlerine göre çok daha büyük ve veri işleme hızı düşük olan DC'lerin işini bitirmesi beklenmez. Bu da son veri değişimde karıştırma veri trafiğini ve işin toplam tamamlanma süresini düşürür.

Sunulan yöntemde deneysel çalışma testleri akademik çalışma ortamlarındaki donanım ve veri boyutu sınırlamalarından dolayı iki fazdan oluşmaktadır. Yani DC'ler arasındaki donanım ve veri boyutları birbirine yakın olarak ele alınmaktadır. Bunların farklı olması halinde, yöntem ikiden fazla fazdan oluşabilir.

Bu çalışmada bant genişliği faktörü varsayılan olarak homojen ele alındı. Gelecekteki çalışmalarda farklı bant genişliklerinin yöntemimiz üzerindeki etkisi değerlendirilebilir. Ayrıca çalışmada sunulan yöntemde, ilerleme durumları için kullanılan eşik değeri, veri boyutu ve işin tamamlanması süresi ile değerlendirilerek belirlenebilir.

6. KAYNAKLAR

[1] P. Zikopoulos ve C. Eaton, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*, 1. bs. New York: McGraw Hill Professional, 2011.

[2] S. Curry, E. Kirda, E. Schwartz, W. H. Stewart, ve A. Yoran. (2024, 12 Mart). *Big Data Fuels Intelligence-Driven Security* [Çevrimiçi]. Erişim: https://silo.tips/download/big-data-fuels-intelligence-driven-security.

[3] S. Sagiroglu ve D. Sinanc, "Big data: A review", içinde *2013 International Conference on Collaboration Technologies and Systems (CTS)*, IEEE, May. 2013, ss. 42-47.

[4] S. Jayaraman. (2024, 12 Mart). *77+ Surreal Big Data Statistics To Map Growth in 2024* [Çevrimiçi]. Erişim: https://www.g2.com/articles/big-data-statistics.

[5] M. Mohsin. (2024, 12 Mart). *10 Google Search Statistics You Need To Know in 2023* [Çevrimiçi]. Erişim: https://www.oberlo.com/blog/google-search-statistics#.

[6] J. Flynn. (2024, 12 Mart). *30+ Instagram Statistics* [Çevrimiçi]. Erişim: https://www.zippia.com/advice/instagram-statistics/.

[7] Akuebionwurichardson. (2024, 12 Mart). *Facebook Messenger Revenue and Growth Statistics* [Çevrimiçi]. Erişim: https://www.usesignhouse.com/blog/facebook-messenger-stats.

[8] Apache Software Foundation. (2024, 12 Mart). *Apache Hadoop* [Çevrimiçi]. Erişim: https://hadoop.apache.org.

[9] A. Vulimiri *vd.*, "WANalytics: Geo-Distributed Analytics for a Data Intensive World", içinde *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, New York, USA, 2015, ss. 1087-1092.

[10] Q. Pu *vd.*, "Low Latency Geo-distributed Data Analytics", *ACM SIGCOMM Computer Communication Review*, c. 45, sayı 4, ss. 421-434, 2015.

[11] Z. Liu, X. Yuan, J. Yuan, J. Zhang, Z. Gu, ve L. Zhang, "Multi-Stage Geo-Distributed Data Aggregation With Coordinated Computation and Communication in Edge Compute First Networking", *Journal of Lightwave Technology*, c. 41, sayı 8, ss. 2289-2300, 2023.

[12] Y. Luo ve B. Plale, "Hierarchical MapReduce Programming Model and Scheduling Algorithms", içinde *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, IEEE, 2012, ss. 769-774.

[13] S. M. Marzuni, A. Savadi, A. N. Toosi, ve M. Naghibzadeh, "Cross-MapReduce: Data Transfer Reduction in Geo-distributed MapReduce", *Future Generation Computer Systems*, c. 115, ss. 188-200, 2021.

[14] D. Le Quoc, C. Fetzer, P. Felber, E. Riviere, V. Schiavoni, ve P. Sutra, "UniCrawl: A Practical Geographically Distributed Web Crawler", içinde *2015 IEEE 8th International Conference on Cloud Computing*, IEEE, 2015, ss. 389-396.

[15] J. Wang, X. Li, R. Ruiz, J. Yang, ve D. Chu, "Energy Utilization Task Scheduling for MapReduce in Heterogeneous Clusters", *IEEE Trans Serv Comput*, c. 15, sayı 2, ss. 931-944, 2022.

[16] S. Imai, C. A. Varela, ve S. Patterson, "A Performance Study of Geo-Distributed IoT Data Aggregation for Fog Computing", içinde *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, IEEE, 2018, ss. 278-283.

[17] W. Xiao, W. Bao, X. Zhu, ve L. Liu, "Cost-Aware Big Data Processing Across Geo-Distributed Datacenters", *IEEE Transactions on Parallel and Distributed Systems*, c. 28, sayı 11, ss. 3114-3127, Kas. 2017.

[18] S. Nithyanantham ve G. Singaravel, "Resource and Cost Aware Glowworm Mapreduce Optimization Based Big Data Processing in Geo Distributed Data Center", *Wirel Pers Commun*, c. 117, sayı 4, ss. 2831-2852, 2021.

[19] T. Gouasmi, W. Louati, ve A. Hadj Kacem, "Geo-Distributed BigData Processing for Maximizing Profit in Federated Clouds Environment", içinde *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, IEEE, 2018, ss. 85-92.

[20] Y. Qin, W. Han, Y. Yang, ve W. Yang, "Joint energy optimization on the server and network sides for geo-distributed data centers", *J Supercomput*, c. 77, sayı 7, ss. 7757-7790, 2021.

[21] J. Wang, X. Li, R. Ruiz, H. Xu, ve D. Chu, "Allocating MapReduce workflows with deadlines to heterogeneous servers in a cloud data center", *Service Oriented Computing and Applications*, c. 14, sayı 2, ss. 101-118, 2020.

[22] Y. Jin, Y. Gao, Z. Qian, M. Zhai, H. Peng, ve S. Lu, "Workload-Aware Scheduling Across Geo-distributed Data Centers", içinde *2016 IEEE Trustcom/BigDataSE/ISPA*, IEEE, 2016, ss. 1455-1462.

[23] W. Wang, W. Zhao, C. Cai, J. Huang, X. Xu, ve L. Li, "An efficient image aesthetic analysis system using Hadoop", *Signal Process Image Commun*, c. 39, ss. 499-508, 2015.

[24] A. Atrey, G. Van Seghbroeck, H. Mora, F. De Turck, ve B. Volckaert, "SpeCH: A scalable framework for data placement of data-intensive services in geo-distributed clouds", *Journal of Network and Computer Applications*, c. 142, ss. 1-14, 2019.

[25] C. Li, C. Zhang, B. Ma, ve Y. Luo, "Efficient multi-attribute precedence-based task scheduling for edge computing in geo-distributed cloud environment", *Knowl Inf Syst*, c. 64, sayı 1, ss. 175-205, 2022.

[26] J. Wang, X. Li, R. Ruiz, H. Xu, ve D. Chu, "Allocating MapReduce workflows with deadlines to heterogeneous servers in a cloud data center", *Service Oriented Computing and Applications*, c. 14, sayı 2, ss. 101-118, Haz. 2020.

[27] V. Pandey ve P. Saini, "A heuristic method towards deadline-aware energy-efficient mapreduce scheduling problem in Hadoop YARN", *Cluster Comput*, c. 24, sayı 2, ss. 683-699, Haz. 2021.

[28] C. Li, J. Tang, ve Y. Luo, "Load Balance Based Job Scheduling in Geo-Distributed Clouds", *Wirel Pers Commun*, c. 107, sayı 1, ss. 169-192, Tem. 2019.

[29] A. Lordache, C. Morin, N. Parlavantzas, E. Feller, ve P. Riteau, "Resilin: Elastic MapReduce over Multiple Clouds", içinde *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, IEEE, 2013, ss. 261-268.

[30] Z. Xiao, H. Chen, ve B. Zang, "A Hierarchical Approach to Maximizing MapReduce Efficiency", içinde *2011 International Conference on Parallel Architectures and Compilation Techniques*, IEEE, 2011, ss. 167-168.

[31] C. Jayalath, J. Stephen, ve P. Eugster, "From the Cloud to the Atmosphere: Running MapReduce across Data Centers", *IEEE Transactions on Computers*, c. 63, sayı 1, ss. 74-87, 2014.

[32] B. Heintz, A. Chandra, R. K. Sitaraman, ve J. Weissman, "End-to-End Optimization for Geo-Distributed MapReduce", *IEEE Transactions on Cloud Computing*, c. 4, sayı 3, ss. 293-306, 2016.

[33] X. Xu *vd.*, "Trading Cost and Throughput in Geo-Distributed Analytics With A Two Time Scale Approach", *IEEE Transactions on Cloud Computing*, c. 10, sayı 3, ss. 2163-2177, 2022.

[34] L. Chen, S. Liu, ve B. Li, "Optimizing Network Transfers for Data Analytic Jobs Across Geo-Distributed Datacenters", *IEEE Transactions on Parallel and Distributed Systems*, c. 33, sayı 2, ss. 403-414, 2022.

[35] Y. Chen, L. Luo, B. Ren, ve D. Guo, "Geo-Distributed IoT Data Analytics With Deadline Constraints Across Network Edge", *IEEE Internet Things J*, c. 9, sayı 22, ss. 22914-22929, 2022.

[36] M. Cavallo, C. Polito, G. Di Modica, ve O. Tomarchio, "H2F:a Hierarchical Hadoop Framework for big data processing in geo-distributed environments", içinde *Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, New York, USA, 2016, ss. 27-35.

[37] L. Wang *vd.*, "G-Hadoop: MapReduce across distributed data centers for data-intensive computing", *Future Generation Computer Systems*, c. 29, sayı 3, ss. 739-

750, 2013.

[38] Guo Zhimao ve Zhou Aoying, “Overview of Data Quality and Data Cleaning Research”, *Journal of Software*, c. 13, sayı 11, ss. 2076-2082, 2002.

[39] P. Gao, Z. Han, ve F. Wan, “Big Data Processing and Application Research”, içinde *2020 2nd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM)*, IEEE, 2020, ss. 125-128.

[40] A. Botta, W. de Donato, V. Persico, ve A. Pescapé, “Integration of Cloud computing and Internet of Things: A survey”, *Future Generation Computer Systems*, c. 56, ss. 684-700, 2016.

[41] B. M. Purcell, “Big data using cloud computing”, *Journal of Technology Research*, c. 5, sayı 1, 2014.

[42] A. Oussous, F.-Z. Benjelloun, A. Ait Lahcen, ve S. Belfkih, “Big Data technologies: A survey”, *Journal of King Saud University - Computer and Information Sciences*, c. 30, sayı 4, ss. 431-448, 2018.

[43] N. Golov ve L. Rönnbäck, “Big Data Normalization for Massively Parallel Processing Databases”, *Comput Stand Interfaces*, c. 54, ss. 86-93, 2017.

[44] A. Gandomi ve M. Haider, “Beyond the Hype: Big Data Concepts, Methods, and Analytics”, *Int J Inf Manage*, c. 35, sayı 2, ss. 137-144, 2015.

[45] E. Aktan, “Büyük Veri: Uygulama Alanları, Analitiği ve Güvenlik Boyutu”, *Bilgi Yönetimi*, c. 1, sayı 1, ss. 1-22, 2018.

[46] E. Dumbill, “Big Data Now Current Perspective”, *O’Reilly Media*, c. 47, ss. 98-115, 2012.

[47] C. Kacfah Emani, N. Cullot, ve C. Nicolle, “Understandable Big Data: A survey”, *Comput Sci Rev*, c. 17, ss. 70-81, 2015.

[48] S. Chandra, S. Ray, ve R. T. Goswami, “Big Data Security: Survey on Frameworks and Algorithms”, içinde *2017 IEEE 7th International Advance Computing Conference (IACC)*, IEEE, 2017, ss. 48-54.

[49] B. Cyganek *vd.*, “A Survey of Big Data Issues in Electronic Health Record Analysis”, *Applied Artificial Intelligence*, c. 30, sayı 6, ss. 497-520, 2016.

[50] Y. Gahi, M. Guennoun, ve H. T. Mouftah, “Big Data Analytics: Security and privacy challenges”, içinde *2016 IEEE Symposium on Computers and Communication (ISCC)*, IEEE, 2016, ss. 952-957.

[51] A. Katal, M. Wazid, ve R. H. Goudar, “Big data: Issues, challenges, tools and Good practices”, içinde *2013 Sixth International Conference on Contemporary Computing (IC3)*, IEEE, 2013, ss. 404-409.

[52] T. D. Thanh, S. Mohan, E. Choi, S. Kim, ve P. Kim, "A Taxonomy and Survey on Distributed File Systems", içinde *2008 Fourth International Conference on Networked Computing and Advanced Information Management*, IEEE, 2008, ss. 144-149.

[53] K. Shvachko, H. Kuang, S. Radia, ve R. Chansler, "The Hadoop Distributed File System", içinde *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, 2010, ss. 1-10.

[54] X. Sun, L. Zhao, J. Chen, Y. Cai, D. Wu, ve J. Z. Huang, "Non-MapReduce computing for intelligent big data analysis", *Eng Appl Artif Intell*, c. 129, ss. 107648, 2024.

[55] School of SRE. (2024, 12 Mart). *Evolution and Architecture of Hadoop* [Çevrimiçi]. Erişim: https://linkedin.github.io/school-of-sre/level101/big_data/evolution.

[56] Apache Software Foundation. (2024, 12 Mart). *HDFS Architecture* [Çevrimiçi]. Erişim: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

[57] K. Shvachko, H. Kuang, S. Radia, ve R. Chansler, "The Hadoop Distributed File System", içinde *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, 2010, ss. 1-10.

[58] R. W. A. Fazul ve P. P. Barcelos, "The HDFS Replica Placement Policies: A Comparative Experimental Investigation", 2022, ss. 151-166.

[59] K. Swaroopa, A. Satya Phani Kumari, N. Manne, R. Satpathy, ve T. Pavan Kumar, "An Efficient Replication Management System for HDFS Management", *Mater Today Proc*, c. 80, ss. 2799-2802, 2023.

[60] M. Saadoon, S. H. Ab. Hamid, H. Sofian, H. H. M. Altarturi, Z. H. Azizul, ve N. Nasuha, "Fault tolerance in big data storage and processing systems: A review on challenges and solutions", *Ain Shams Engineering Journal*, c. 13, sayı 2, s. 101538, Mar. 2022.

[61] M. A. Ahmed, M. H. Khafagy, M. E. Shaheen, ve M. R. Kaseb, "Dynamic Replication Policy on HDFS Based on Machine Learning Clustering", *IEEE Access*, c. 11, ss. 18551-18559, 2023.

[62] M. Maurya ve S. Mahajan, "Performance analysis of MapReduce programs on Hadoop cluster", içinde *2012 World Congress on Information and Communication Technologies*, IEEE, Eki. 2012, ss. 505-510.

[63] K. Shvachko, H. Kuang, S. Radia, ve R. Chansler, "The Hadoop Distributed File System", *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, Incline Village, NV, USA, 2010, pp. 1-10, 2010.

[64] Y. Zhang, L. Cui, W. Wang, ve Y. Zhang, "A survey on software defined

networking with multiple controllers", *Journal of Network and Computer Applications*, c. 103, ss. 101-118, 2018.

[65] K. Kalia ve N. Gupta, "Analysis of hadoop MapReduce scheduling in heterogeneous environment", *Ain Shams Engineering Journal*, c. 12, sayı 1, ss. 1101-1110, 2021.

[66] J. Dean ve S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *Commun ACM*, c. 51, sayı 1, ss. 107-113, Oca. 2008.

[67] C. Şener. (2024, 12 Mart). *MapReduce Tekniğine Giriş* [Çevrimiçi]. Erişim: https://indico.truba.gov.tr/event/52/contributions/434/attachments/97/239/MapReduce%20Teknig%CC%86ine%20Giris%CC%A7.pdf.

[68] G. S. Sadasivam ve G. Baktavatchalam, "A Novel Approach to Multiple Sequence Alignment using Hadoop Data Grids", *Int J Bioinform Res Appl*, c. 6, sayı 5, s. 472, 2010.

[69] J. Ekanayake, S. Pallickara, ve G. Fox, "MapReduce for Data Intensive Scientific Analyses", içinde *2008 IEEE Fourth International Conference on eScience*, IEEE, 2008, ss. 277-284.

[70] B. Schölkopf, J. Platt, ve T. Hofmann, "Map-Reduce for Machine Learning on Multicore", içinde *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, MIT Press, 2007, ss. 281-288.

[71] Cloudera. (2024, 12 Mart). *Hadoop Developer Training* [Çevrimiçi]. Erişim: http://www.cloudera.com/wpcontent/uploads/2010/01/1-ThinkingAtScale.pdf.

[72] I. Polato, R. Ré, A. Goldman, ve F. Kon, "A comprehensive view of Hadoop research—A systematic literature review", *Journal of Network and Computer Applications*, c. 46, ss. 1-25, 2014.

[73] A. El Yazidi, M. S. Azizi, Y. Benlachmi, ve M. L. Hasnaoui, "Apache Hadoop-MapReduce on YARN framework latency", *Procedia Comput Sci*, c. 184, ss. 803-808, 2021.

[74] S. Kim. (2024, 12 Mart). *YARN (MapReduce 2)* [Çevrimiçi]. Erişim: https://sungsoo.github.io/2013/12/12/yarn-mapreduce2.html.

[75] S. Çetinkaya, "Hadoop/MapReduce Teknolojisi Kullanılarak Hızlı Tüketim Sektöründe Büyük Veri Analizi", Yüksek Lisans Tezi, Bilgisayar Mühendisliği Ana Bilim Dalı, Fen Bilimleri Enstitüsü, İstanbul Üniversitesi, İstanbul, Türkiye, 2016.

[76] Apache Software Foundation. (2024, 12 Mart). *Apache Hadoop YARN* [Çevrimiçi]. Erişim: https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html.

[77] Cloudera. (2024, 12 Mart). *Major changes when migrating to MapReduce 2*

[Çevrimiçi]. Erişim: https://docs.cloudera.com/cdp-private-cloud-upgrade/latest/upgrade-cdh/topics/yarn-major-changes-mrv2.html?.

[78] C. Jayalath ve P. Eugster, "Efficient Geo-distributed Data Processing with Rout", içinde *2013 IEEE 33rd International Conference on Distributed Computing Systems*, IEEE, 2013, ss. 470-480.

[79] Y. Luo, Z. Guo, Y. Sun, B. Plale, J. Qiu, ve W. W. Li, "A Hierarchical Framework for Cross-Domain MapReduce Execution", içinde *Proceedings of the second international workshop on Emerging computational methods for the life sciences*, New York, USA, 2011, ss. 15-22.

[80] F. Galton, "I. Family likeness in stature", *Proceedings of the Royal Society of London*, c. 40, ss. 242-245, 1886.

[81] M. B. Kaya, "Deming Regresyon ve Passing-Bablok Regresyon Yöntemlerinde Jackknife Yöntemi", Yüksek Lisans Tezi, Ekonometri Ana Bilim Dalı, Sosyal Bilimler Enstitüsü, Sivas Cumhuriyet Üniversitesi, Sivas, Türkiye, 2023.

[82] D. Atasoy, "Optimizasyon ve Regresyon Modelleri ile Meteorolojik Tahminleme Denemesi", Doktora Tezi, Matematik Ana Bilim Dalı, Fen Bilimleri Enstitüsü, Iğdır Üniversitesi, Iğdır, Türkiye, 2023.

[83] B. G. Tabachnick ve L. S. Fidell, *Using multivariate statistics (5th ed)*. New York: Pearson Education. 2007.

[84] V. Barnett ve T. Lewis, *Outliers in Statistical Data*, 3. bs. California: John Wiley Sons. Academic Publishers, 1994.

[85] P. J. Rousseeuw ve A. M. Leroy, *Robust Regression and Outlier Detection*, Wiley, 1987. doi: 10.1002/0471725382.

[86] B. Kafkas, "Boylamsal Verilerde Çok Düzeyli Doğrusal Regresyon ve Kantil Regresyon Yöntemlerinin Karşılaştırılması", Doktora Tezi, Eğitim Bilimleri Ana Bilim Dalı, Eğitim Bilimleri Enstitüsü, Hacettepe Üniversitesi, Ankara, Türkiye, 2023.

[87] J. Fan ve I. Gijbels, *Local Polynomial Modelling and Its Applications*, Routledge, 2018. doi: 10.1201/9780203748725.

[88] J. O. Rawlings, S. G. Pantula, ve D. A. Dickey, *Applied Regression Analysis*, New York: Springer-Verlag, 1998. doi: 10.1007/b98890.

[89] B. Varol, "Parçalı Regresyon ile Polinom Regresyon Analizlerinin Karşılaştırılması", Yüksek Lisans Tezi, Biyoistatistik Ana Bilim Dalı, Sağlık Bilimleri Enstitüsü, Adnan Menderes Üniversitesi, Aydın, Türkiye, 2017.

[90] L. S. Aiken, S. G. West, S. C. Pitts, A. N. Baraldi, ve I. C. Wurpts, "Multiple Linear Regression", içinde *Handbook of Psychology, Second Edition*, Wiley, 2012. doi: 10.1002/9781118133880.hop202018.

[91] G. Sever, "İstatistiksel Daraltıcı Yöntemlerden Ridge Regresyon, Lassa Regresyon ve Elastik Net Regresyonun Tahminleme ve Sınıflandırma Performanslarının Karşılaştırılması", Yüksek Lisans Tezi, Biyoistatistik Ana Bilim Dalı, Sağlık Bilimleri Enstitüsü, Eskişehir Osmangazi Üniversitesi, Eskişehir, Türkiye, 2021.

[92] D. Alakaya, "Kantil Regresyon ve Doğrusal Regresyon Yöntemlerinin Performansını Etkileyen Faktörlerin İncelenmesi", Yüksek Lisans Tezi, Biyoistatistik ve Tıbbi Bilişim Ana Bilim Dalı, Sağlık Bilimleri Enstitüsü, Mersin Üniversitesi, Mersin, Türkiye, 2019.

[93] L. Hamilton, *Regression with Graphics: A Second Course in Applied Statistics*, 1. bs. Duxbury Press, 1992.

[94] F. Ahmad. (2024, 12 Mart). *Puma Dataset* [Çevrimiçi]. Erişim: https://engineering.purdue.edu/~puma/datasets.htm.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Emin ŞEŞEN

Yabancı Dili : İngilizce

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Doktora	Bilgisayar Müh.	Düzce Üniversitesi	2024
Y. Lisans	Bilgisayar Müh.	Düzce Üniversitesi	2017
Lisans	Bilgisayar Öğretmenliği	Abant İzzet Baysal Üniversitesi	2009
Lise	Fen Bilimleri	Denizli Lisesi	2005

TEZDEN ÇIKAN YAYINLAR

1. E. Şeşen, S. Kırışoğlu and R. Kara, “GSelf-MapReduce: A Method For Enhancing MapReduce Performance in Distributed Heterogeneous Data Centers”, *IEEE Access*.