DİNAMİK ANAHTAR GRUPLAMA: DAĞITIK AKAN-VERİ KATARI İŞLEME SİSTEMLERİ İÇİN BİR YÜK DENGELEME ALGORİTMASI

DYNAMIC KEY GROUPING: A LOAD BALANCING ALGORITHM FOR DISTRIBUTED STREAM PROCESSING ENGINES

ORHUN DALABASMAZ

DOÇ. DR. AHMET BURAK CAN
Tez Danışmanı

Hacettepe Üniversitesi
Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin
Bilgisayar Mühendisliği Anabilim Dalı için Öngördüğü
YÜKSEK LİSANS TEZİ olarak hazırlanmıştır.

2018

ORHUN DALABASMAZ'ın hazırladığı **"Dinamik Anahtar Gruplama: Dağıtık Akan-veri Katarı İşleme Sistemleri İçin Bir Yük Dengeleme Algoritması"** adlı bu çalışma aşağıdaki jüri tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Prof. Dr. Ali Aydın SELÇUK
Başkan .....

Doç. Dr. Ahmet Burak CAN
Danışman .....

Doç. Dr. Sevil Şen AKAGÜNDÜZ
Üye .....

Doç. Dr. Hasan Şakir BİLGE
Üye .....

Yrd. Doç. Dr. Murat AYDOS
Üye .....

Bu tez Hacettepe Üniversitesi Fen Bilimleri Enstitüsü tarafından **YÜKSEK LİSANS TEZİ** olarak onaylanmıştır.

Prof. Dr. Menemşe GÜMÜŞDERELİOĞLU
Fen Bilimleri Enstitüsü Müdürü

Aileme…

YAYINLAMA VE FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezimin/raporumun tamamını veya herhangi bir kısmını, basılı (kağıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma iznini Hacettepe üniversitesine verdiğimi bildiririm. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları bana ait olacaktır.

Tezin kendi orijinal çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanması zorunlu metinlerin yazılı izin alarak kullandığımı ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

☒ **Tezimin/Raporumun tamamı dünya çapında erişime açılabilir ve bir kısmı veya tamamının fotokopisi alınabilir.**
(Bu seçenekle teziniz arama motorlarında indekslenebilecek, daha sonra tezinizin erişim statüsünün değiştirilmesini talep etseniz ve kütüphane bu talebinizi yerine getirse bile, tezinin arama motorlarının önbelleklerinde kalmaya devam edebilecektir.)

☐ **Tezimin/Raporumun ..... tarihine kadar erişime açılmasını ve fotokopi alınmasını (İç Kapak, Özet, İçindekiler ve Kaynakça hariç) istemiyorum.**
(Bu sürenin sonunda uzatma için başvuruda bulunmadığım taktirde, tezimin/raporumun tamamı her yerden erişime açılabilir, kaynak gösterilmek şartıyla bir kısmı ve ya tamamının fotokopisi alınabilir)

☐ **Tezimin/Raporumun ..... tarihine kadar erişime açılmasını istemiyorum, ancak kaynak gösterilmek şartıyla bir kısmı veya tamamının fotokopisinin alınmasını onaylıyorum.**

☐ **Serbest Seçenek/Yazarın Seçimi**

18 / 01 / 2018

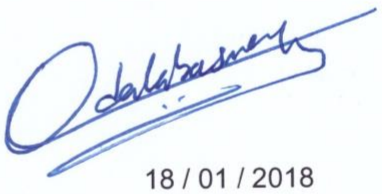
ORHUN DALABASMAZ

ETİK

Hacettepe Üniversitesi Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında,

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversitede veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.



18 / 01 / 2018

ORHUN DALABASMAZ

ÖZET

DİNAMİK ANAHTAR GRUPLAMA: DAĞITIK AKAN-VERİ KATARI İŞLEME SİSTEMLERİ İÇİN YÜK DENGELEME ALGORİTMASI

Orhun DALABASMAZ

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Danışmanı: Doç. Dr. Ahmet Burak CAN

Ocak 2018, 133 sayfa

İçinde bulunduğumuz teknoloji çağıyla beraber teknolojik cihazlar yaşamımızın vazgeçilmez birer parçası olmuştur. Her gün, kullanılan cihazlar, bu cihazlara yazılan uygulamalar ve kullanıcılar da artmaktadır. Bütün bu artışlar, üretilen veri miktarındaki çeşitlilik ve hacim artışını da beraberinde getirmiştir. Üretilen verinin hacmi ve çeşitliliği öyle artmıştır ki, artık tek bir makinenin tek başına altından kalkabilmesine imkân kalmamıştır. Bununla birlikte, gelişen ihtiyaçlar verilerin anlık olarak işlenerek gerçek zamanlı sonuç üretebilmelerini gerekmektedir. Bu kapsamda, bir işi yapabilmek için bilgisayar kümeleri kullanılmaktadır. Verilerin bu kümelere dağıtılarak en kısa sürede işlenerek çıktı üretebilmesini sağlamak amaçlanmaktadır. Bunun sağlanabilmesi içinse, yükün, yani verilerin, kümedeki bütün makinelere mümkün olduğunca dengeli dağıtılması gerekmektedir. Yükün makinelere dengesiz dağıtılması, birtakım makinelerin diğerlerinden daha yoğun çalışacağı ve böylece her makinenin verimli

kullanılamayacağı anlamına gelmektedir. Verimliliğin düşmesi, sistemin çalışma süresinin artması ve üretkenliğin azalması anlamına gelmektedir. Bu da gerçek zamanlı sonuç üretemeyeceğimiz anlamına gelmektedir. Yükün makinelere dengeli dağıtılması, verinin içeriğine doğrudan bağlıdır. Veri ne kadar homojen dağılıma sahipse, yük o kadar dengeli dağıtılırken, ne kadar çarpık gelirse o kadar dengesiz dağıtılacaktır. Veriler arası bağlamın olmadığı durumda en iyi yük dağılımını sağlayan Karışık Gruplama (SG) yöntemi kullanılırken, veriler arası bağlamın olduğu durumda ise, verinin içeriğine bağlı olarak çalışan Anahtar Gruplama (KG) yöntemi kullanılmaktadır. SG yönteminde her veri makinelere rastgele dağıtılırken, KG yönteminde her veri karma değerine göre bir makineye atanmaktadır. Bu sayede durumlu verilerin tek bir makinede toplanması ve tek bir sonuç çıkartılması sağlanmaktadır. Ancak KG yöntemi, bir verinin çarpık gelmesi durumunda yükün tek bir makinede yoğunlaşmasını sağlayarak sistemin verimli çalışamamasına sebebiyet vermektedir. Parçalı Anahtar Gruplama (PKG) yöntemi ise her veri için iki karma değerinin hesaplanmasını, yani yükün iki makineye dağıtılmasını sağlamaktadır. Bu sayede çarpık gelen verilerde dahi sistemin performansında gözle görülür iyileşmeler elde edilebilmektedir. Ancak bu yöntem de bazı verilerin çok yoğun geldiği durumlarda sistemin verimsiz olmasına sebebiyet vermektedir. Çünkü yük her ne kadar iki makineye dağıtılsa da çok yoğun gelen veri karşısında yükü sadece iki makineye dağıtabildiğinden, diğer makinelerin yükü daha az olmakta, böylece verimsizlik ve performans kayıpları yaşanabilmektedir. Ayrıca bu yöntemle, herhangi bir veri için hesaplanan iki karma değerinin birbirinden farklı olacağı garanti edilememektedir. Tüm bu koşullarda sistemin verimliliğinin ve performansının, verinin içeriğinden bağımsız olarak sürekli yüksek olması gerekliliği doğmaktadır. Bu çalışma kapsamında, yükün makinelere veri içeriğinden bağımsız olarak her zaman dengeli dağıtılabilmesi için Dinamik Anahtar Gruplama (DKG) yöntemi önerilmiştir. Bu yöntem ile yoğun gelen verilerin tespiti yapılmakta ve yoğun verilerin daha çok makineye dağıtılabilmesi sağlanmaktadır. Bu sayede, özellikle çarpık verilerin çok olduğu durumlarda, sistemin üretkenliğinde ve çalışma zamanında iyileşmeler gözlemlenmiş ve oldukça başarılı sonuçlar alınmıştır.

Anahtar Kelimeler: akan-veri katarı yönlendirme, dağıtık akan-veri katarı işleme, iki seçeneğin gücü, anahtar gruplama, yük dengeleme.

ABSTRACT

DYNAMIC KEY GROUPING: A LOAD BALANCING ALGORITHM FOR DISTRIBUTED STREAM PROCESSING ENGINES

Orhun DALABASMAZ

Master of Science, Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Ahmet Burak CAN

January 2018, 133 pages

With the technological age we are in, technological devices are an indispensable part of our life. Every day, variety of devices, applications and number of users are also increasing. All these increases cause a huge grow of data produced and so the variety of data. The volume and the variety of the produced data is so increased that it is no longer possible for single machine to handle alone. On the other hand, requirements force us to process data in real-time. Therefore, cluster of machines is used for high efficiency, fault-tolerant and robust systems. By using a cluster, we aim to process all data as soon as possible by distributing the data to all nodes in the cluster. In order to achieve this, the data or the load should be distributed to the machines as equally as possible. Unbalanced distribution of the load means that a number of machines will work more intensively than others, and thus each machine will not be used efficiently. Reduced productivity and efficiency leads the increase of latency and decrease of throughput. Therefore, the system cannot produce real-time results. In such systems,

balancing the load to the machines is directly connected to the contents of the data. The more homogeneous the data, the more balanced the load is, the more skewed it will be distributed unevenly. Shuffle Grouping (SG) is the best option when there is no relation between the data incoming. On the other hand, when there is a relation between the incoming data, the best option is Key Grouping (KG) which assigns the incoming data to the target machines by examining the data content. While the data is distributed to machines randomly by using Round-Robin with Shuffle Grouping, the data is distributed to machines by calculation a hash value for each data with Key Grouping. Therefore, related data can be gathered in the same machine and there is no need to aggregate from several machines. However, both of the grouping methods may become useless and inefficient depending on the data. When there is a relation between the data incoming, Shuffle Grouping cannot be used efficiently, because Shuffle Grouping does not care about the content of incoming data. On the other hand, when the data is skewed, Key Grouping routes the skewed data to the only one machine and most of the load will be concentrated on single machine. In other words, the more the data is skewed, the more inefficient load balancing occurs. This also leads inefficiency, more latency, less throughput and non-real-time results. Partial Key Grouping (PKG), on the other hand, specifies two target machines by calculating two different hash values and chooses the less loaded one for efficient load balancing. With Partial Key Grouping, every data can be distributed to two different machines. Even if the data is slightly skewed, system may have better performance and better load balancing than Key Grouping. However, if the data is so skewed and some of the data is recurring so many times, even Partial Key Grouping may show bad performance. Highly skewed load would be distributed to only two machines, thus, other machines in the cluster would have far less load to process and this leads to inefficiency and performance issues. On the other hand, this method, Partial Key Grouping, does not guarantee the two calculated hash values for one data will be different. In all these conditions, the efficiency and performance of the system must be consistently high regardless of the content of the data. In this study, Dynamic Key Grouping (DKG) method is proposed to distribute the load to the machines at all times regardless of the data content. With this method, skewed data are detected and can be distributed to

more servers. Moreover, improvements were observed in the throughput and latency of the system, especially when the data is highly skewed and very successful results were obtained.

Keywords: distributed stream processing, load balancing, power of multiple choices, key grouping, stream partitioning.

TEŞEKKÜR

Çalışma konumu seçtiğimde beni desteklediği ve akademik vizyonu ile bana yol gösterdiği için danışmanım Sayın Ahmet Burak Can'a, çalışmam kapsamında gerçekleştirdiğim deneylerimi yapabilmem için Canavar adını verdiğim profesyonel iş sunucusunu bana verdiği için sevgili ağabeyim Sayın Ozan Ali Kaya'ya ve başta eşim Tuğçe Dalabasmaz olmak üzere beni bu uzun ve yorucu yolda hep destekleyerek yanımda olan aileme teşekkür ediyorum.

İÇİNDEKİLER

Sayfa

SİMGELER ve KISALTMALAR

Simgeler

Lavg	Ortalama Yük
Tm	Hedef Makine
Nm	Tercih Edilebilecek Makine Sayısı
L	Makinenin yükü
Li	İdeal yük değeri
Lt	Eşik yük değeri
Th	Üretkenlik değeri
Lsd	Standart Sapma değeri
Ldc	Dağıtım maliyet değeri

Kısaltmalar

AVİ	Akan-veri İşleme
SPE	Stream Processing Engines
RDD	Resilient Distributed Datasets
DAG	Directed Acyclic Graph
SG	Shuffle Grouping
KG	Key Grouping
PKG	Partial Key Grouping
DKG	Dynamic Key Grouping
CMR	Correlation-aware Multi-route Stream Query Optimizer

JVM	Java Virtual Machine
GC	Garbage Collection
CEP	Complex Event Processing
PE	Processing Element
PEI	Processing Element Instance
CP	Consistent Grouping
DAG	Directed Acyclic Graph
3V, 5V, 7V	Volume, Velocity, Variety, Veracity, Value, Variability, Visualization
KB	Kilobytes
MB	Megabytes (10^3 KB)
GB	Gigabytes (10^3 MB)
TB	Terrabytes (10^3 GB)
PB	Pettabytes (10^3 TB)
EB	Exabytes (10^3 PB)
ZB	Zettabytes (10^3 EB)
YB	Yottabytes (10^3 ZB)

Kavramlar

anahtar	key
anahtar alanı	key space
anahtar birimi	key item
anahtar bölüntüleme	key splitting
anahtar gruplama	key grouping

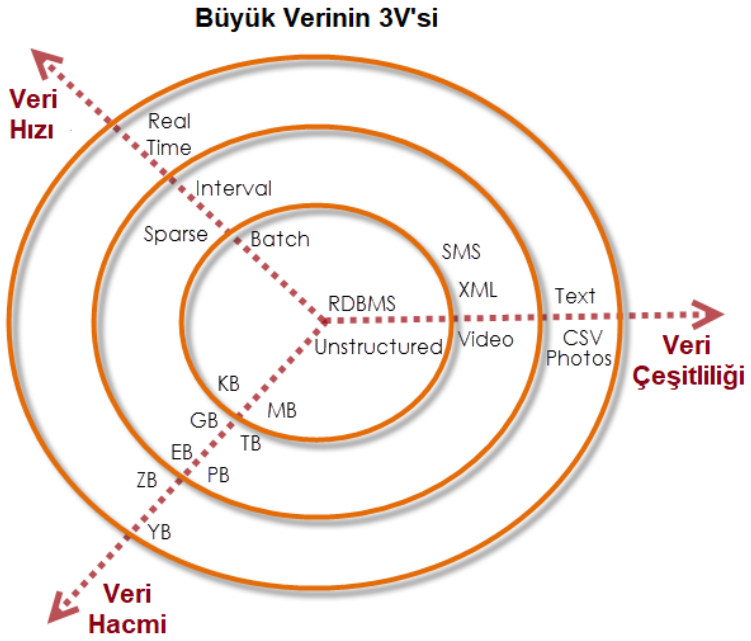
akan-veri	stream / streaming data
atık toplama döngüsü	garbage collection
ayrıştırıcı birim	splitter
bağlılaşım	correlation
bölüntü	partition
bölüntüleme	partitioning
bütünleştirmek	aggregate
bütünleştirme maliyeti	aggregation cost
bütünleştirici birim	aggregator
çalışma günlüğü	log
çok-öğeli	tuple
dağıtım maliyeti	distribution cost (a.k.a. aggregation cost)
dikey büyüme	scale up
dinamik anahtar gruplama	dynamic key grouping
dizin	index
durumlu	stateful
durumsuz	stateless
düzgelenmiş	normalized
eniyileyici	optimization
geçirgeç	transistör
görev	task
gözlemci birim	distribution observer
hızlı çakan	fail-fast

icracı	executor
ilinge	topology
ilk alan	baby space
iş parçası	thread
iş süreci	process
iş takipçisi	job tracker
iş yapan birim	processing element
işçi birim	worker node / bolt
işçi makine	worker
işleç	operator
işlem öğesi	processing element instance
gecikme	latency
genç alan	young generation
java sanal makinesi	java virtual machine
kalıcı alan	permanent generation
karışık gruplama	shuffle grouping
karma	hash
karmaşık olay verisi işleme	complex event processing
kaynak birim	spout
kimlik bilgisi	ID
konu	topic
küçük-veri-kümeleri	micro-batching
olay-verisi	event data

olgunlaşmış alan	old generation
orta alan	teenage space
parçalı anahtar gruplama	partial key grouping
son alan	old Space
standart sapma	standard deviation
tutarlı gruplama	consistent grouping
usta birim	master node
uyarlanabilen	adaptive
üstveri	metadata
yaşayan alan	survivor space
yatay büyüme	scale out
yatay küçülme	scale down
yerel yük	local load
yönetici birim	supervisor
yönlü döngüsüz çizgeler	directed acyclic graph

1. GİRİŞ

Her gün sayıları katlanarak artan teknolojik cihazlar çevremizi sarmış durumdadır. Hem teknolojideki büyümenin, hem de her gün kullandığımız cihazların sayısının artışı kaçınılmazdır. Bu cihazlar sadece cebimize girmekle kalmamış, aynı zamanda hayatımıza girerek yaşamımızın vazgeçilmez birer parçaları olmuşlardır. Bilgisayarlar, akıllı cep telefonları, akıllı kol saatleri, akıllı mutfak eşyaları ve akıllı evler bunlardan bazılarıdır. Daha önce hayatımızda yer alan her araç ve cihaz, teknoloji ile "akıllanarak" hayatımızdaki yerini daha da güçlendirmiştir. Öyle ki, günümüzde artık teknolojinin girmediği ve akıllandırmadığı çok az şey vardır. Hayatımızın her alanında var olan bu dijital hareketler, daha çok verinin daha hızlı üretildiği, üretilen veri kaynağının ve dolayısıyla çeşidinin de arttığı anlamına gelmektedir.



Şekil 1. Büyük Verinin 3Vs [1]

Artan veri çeşidi, hızı ve miktarı; verilerin depolanması ve işlenmesi problemlerini de beraberinde getirmektedir. Tüm bu problemler Büyük Veri kapsamına girmektedir. Büyük Veri kavramı ilk olarak 2005 yılında O'Reilly Media'da çalışan Roger Mougalas

tarafından kullanılmıştır. Peki, Büyük Veri denilince aklımıza ne gelmeli? Büyük Veri tanımı yaparken kesin ve kati sınırlar çizemeyiz. Büyük Veri derken, genel karakteristik yapılarının bir veya birkaçına sahip olan bir oluşumdan bahsediyoruz demektir. Gartner analisti Doug Laney'in 2001 yılında yayınladığı çalışmasında [1] da bahsettiği ve günümüzde halen genel kabul gören bu karakteristiklerden bazıları Veri Hacmi, Veri Hızı ve Veri Çeşitliliğidir. Bunlara ek olarak Veri Doğruluğu, Veri Değeri, Veri Kararsızlığı ve Veri Görselleştirilmesi de son zamanlarda sıklıkla dile getirilen diğer karakteristiklerdir.

3V, 5V veya 7V olarak anılan bu karakteristikleri kısaca tanıtacak olursak;

1. **Veri Hacmi**

   Üretilen veri miktarının büyüklüğünü ifade eder. Eskiden gigabyte (GB) birimi ile ifade edilen veri hacimleri artık zettabyte (ZB) ve hatta yottabyte (YB) birimleri ile ifade edilmektedir. Nesnelerin İnterneti ile birlikte veri üretimi üssel olarak artış göstermektedir. Örneğin, uluslararası yolcu taşıyan bir uçağın her iki motorundan yılda yaklaşık olarak 2.5 ZB (2,499,841,200 terabyte) büyüklüğünde veri toplanmaktadır.

2. **Veri Hızı**

   Birim zamanda üretilen veri miktarını ifade etmektedir. Örneğin, her dakika, yaklaşık 200+ milyon e-posta gönderilmekte, 4+ milyon Facebook beğeni ve 1+ milyon Instagram beğeni işlemi yapılmaktadır.

3. **Veri Çeşitliliği**

   Üretilen verilerin yapısındaki çeşitliliği ifade etmektedir. Üretilen verilerin bir kısmı metin tabanlı olurken, bir kısmı ses, bir kısmı ise görüntü ve bir kısmı video formatında olabilmektedir. Bütün bu veriler de kendi içlerinde JSON/XML/HTML/PDF/DOC, MP3/WAV, JPEG/PNG, MP4/MPEG/AVI gibi farklı formatlara sahip olabilmektedir.

4. **Veri Doğruluğu**

   Üretilen her verinin işleme alınamayacağını ifade etmektedir. Bazı veriler anlamsız olabilirken bazıları ise bozuk alıcı kaynaklı eksik bilgiler içerebilmektedir. Bu veriler sonucu olumsuz etkileyeceğinden tespit edilip ayıklanmalıdır.

5. **Veri Değeri**

   Veri değeri, büyük verinin temelini ve aynı zamanda amacını oluşturmaktadır. Bütün büyük veri işlemleri, verinin içindeki değeri en hızlı ve doğru biçimde ortaya çıkarmayı hedeflemektedir. Toplanan verileri değere dönüştüremedikten sonra bütün süreç ve yetenekler anlamını yitirmektedir. Çünkü toplanan verilerden çıkartılacak değerler, alınacak aksiyonları ve atılacak adımları belirlemektedir.

6. **Veri Tutarlılığı**

   Veri tutarlılığı, veri çeşitliliğinden farklıdır. Örneğin, bir kahve dükkanında 6 farklı kahve üretiliyor olabilir. Ancak her gün aynı kahveyi satın almanıza rağmen farklı tatlar alıyorsanız bu tutarsızlıktır. Aynısı veri için de geçerlidir. Eğer toplanan verinin anlamı değişim gösteriyorsa bu verinin homojenliğini ve veriden üretilen anlam ve değeri de etkilemektedir.

7. **Veri Görselleştirilmesi**

   Verinin toplanması ve işlenmesi kadar, çıkartılan sonuçların en etkili şekilde okunabilmesi ve yorumlanabilmesi de çok önemlidir. Sonuç olarak verilerin çıktısı da yine başka veriler olmakta ve bu verilerde insanlar tarafından okunamayacak kadar büyük ve kapsamlı olabilmektedir. Bu kapsamda veriler görselleştirilerek, büyük ve karmaşık verilerden grafikler ve tablolar ile daha kolay okunabilecek çıktılar üretilebilmektedir.

Büyük Veriden bahsedebilmek için, saydığımız bu karakteristiklerden bir veya birkaçına sahip olmamız gerekmektedir. Görüldüğü üzere, büyük veri sadece veriyi saklama ve işleme değil aynı zamanda veriyi çözümleyerek bilgi ve veri çıkarsama konularıyla da ilgilenmektedir.

1.1. Problemin Tanımı

Yıllar içerisinde artan veri miktarı ile büyük veri üzerine çalışmalar yapılmaya başlandı ve 2005 yılında Yahoo, Hadoop adını verdiği açık kaynak kodlu büyük veri kütüphanesini tanıttı. Hadoop, Google'ın MapReduce mimarisini kullanarak dağıtık eş zamanlı veri işleme yapan ve yüksek çıktı üreten bir büyük veri kütüphanesidir. Hadoop ile bütün web'in dizinlenmesi amaçlanmıştı. Bugünlerde ise birçok kurum ve şirket tarafından büyük hacimli verileri işlemek için kullanılmaktadır.

Hadoop, offline veri işleme üzerine geliştirilen mimarisi ile çok başarılı olmasına rağmen gerçek zamanlı çıktı üretilmesi gereken birçok durumda etkisiz kalmaktadır. Offline veri işlenirken genellikle süre ve hız göreceli olarak çok önemli olmazken, gerçek zamanlı veri işlemede, verinin sisteme varışından sonra belli bir sürede çıktı üretilmesi beklenmektedir. Uçak motorlarının sağlık durumları, kredi kartı dolandırıcılığı, virüs tespiti, donanım hataları tespiti gibi birçok durumda verinin çok hızlı işlenerek uyarıların en kısa sürede verilmesi hayati önem taşımaktadır. Böylece, durumlara gerçek zamanlı müdahale edilebilirse zarar en aza indirgenmiş olacaktır.

Verilerin geliş hızı her zaman veri işleme hızının altında tutulmalıdır. Aksi takdirde veriler kuyrukta birikmeye başlayacak, sonuç üretiminde gecikmeler yaşanacak ve gecikme süresi gittikçe artacaktır. Bu tür durumlarda sistemin sağlıklı çalışabilmesini sağlamak için sistemlerin dikey ve/veya yatay büyüyebilmesi yani sistemin ölçeklenebilir olması gerekmektedir. Dikey büyüme CPU, RAM gibi fiziksel kaynakların kapasitelerinin yükseltilmesi anlamına gelirken; yatay büyüme çalışan makine kümelerine yeni bir makine eklemek anlamına gelmektedir.

Gerçek zamanlı büyük veri işleyecek sistemler tasarlanırken olası iş yükü tahmin edilmekte ve buna göre kaynak ayırımı yapılmaktadır. Bununla birlikte sistemlerin çalışma anında, sistemi durdurmadan, dikey büyümesini sağlamak pek mümkün olmamaktadır. Bunun yerine yatay büyüme tercih edilmektedir. Böylelikle, ihtiyaç duyulduğunda fiziksel sınırı olan dikey büyüme yerine, teorik olarak sınırı olmayan yatay büyüme tercih edilmektedir. Yatay büyüme, avantajlarının yanı sıra birtakım dezavantajlara da sahiptir. Gerek kümeye eklenecek her bir makinenin ayrı bir maliyet getirmesinden, gerekse dağıtık işlem yapılırken bütün makinelere dağıtılan verilerin

işlendikten sonra tutarlı tek bir sonuç elde edebilmek adına toplanması sırasında oluşan ek maliyetten dolayı, veri işleme aşamasında çalışacak algoritmaların da iyileştirilmesi gerekmektedir.

Gerçek zamanlı veri işleyen sistemlerin dikkate alması gereken iki temel problem vardır: i) veri işleme hızının, veri geliş hızından daha yüksek olması ve ii) verilerin kümedeki bütün makineler arasında dengeli dağıtılması. Veri işleme hızının veri geliş hızından yüksek olması bütün gerçek zamanlı veri işleme uygulamaları için bir zorunluluk olmakla birlikte, doğrudan veri üzerinde çalıştırılacak olan algoritmaların yaptığı iş ile ilgilidir. Bu nedenle bu konuda genel geçer bir çözüm önermek çok da mümkün olamamaktadır.

Verilerin kümedeki makineler arasında dengeli dağıtılabilmesi ise verilerin içeriğine bağlı olmakla birlikte, kabul görmüş yaklaşımlarla iyileştirilebilmektedir. İdeal senaryoda, veriler makinelere eşit şekilde dağıtılabilir ve böylece en kısa çıktı süresini elde etmiş oluruz. Bu noktada bizim için belirleyici olan, işlenecek verilerin aralarında ilişki olup olmadığıdır. Eğer veriler birbirlerinden bağımsız ise, bahsedildiği gibi veriler eşit şekilde dağıtılabilir. Ancak, veriler arasında bir ilişki varsa, yani bir veri tek başına anlamsız veya eksik ise, yani değer üretebilmek için diğer verilere de ihtiyaç duyuyorsa, durumlu verilerin bir araya getirecek bir yapı tasarlamamız gerekmektedir. Geleneksel yaklaşımlar bu probleme iki farklı çözüm önermektedir: i) ilişkili her verinin bir anahtar değeri ile ayırt edilerek aynı makineye gönderilmesi sağlanır, ii) bütün veriler makinelere eşit olarak dağıtılır ama sonrasında fazladan bir makine bu verileri toplama görevini üstelenir ve verileri birleştirerek durumlu verilerden tek bir sonuç üretilmesini sağlar. Birinci yaklaşımda, gelen verilerin içeriğine bağlı olarak, çarpık verilerin çok olması durumunda bazı makinelerde aşırı yük oluşurken bazı makinelerde ise çok az yük oluşabilmektedir. Bu da genele baktığımızda veri işleme hızının oldukça azalmasına ve makinelerin verimli kullanılmamasına sebep olmaktadır. İkinci yaklaşımda ise, yükün her zaman dengeli dağıtılacağı garanti edilmesine rağmen, birbirleriyle ilişkili gelen verilerin bütün makinelere dağılması sonucu, verilerin tek başlarına anlamlı olmamaları ve başka bir makinede tekrar toplanması zorunluluğu doğmaktadır. Bu da sisteme ek yük ve maliyet getirmektedir. Her iki yaklaşım da işlenen

verinin türüne ve içeriğine göre farklı sonuçlar doğurabilmektedir. Gerçek zamanlı veri işlerken, gelecek verilerin türünü ve içeriğini bilemeyeceğimiz için her iki yöntem de pratikte uygulanabilir değildir.

1.2. Çalışmanın Kapsamı

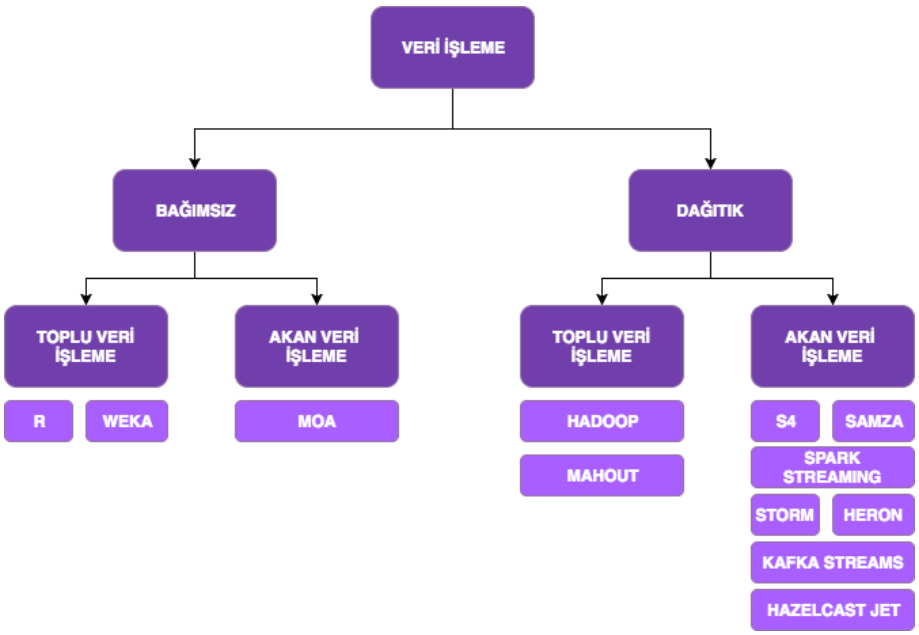
Bu çalışma kapsamında veri yükünün makineler arasında dengeli dağıtılabilmesi problemi işlenmiş olup, mevcutta bulunan Anahtar Gruplama (KG), Karışık Gruplama (SG), Parçalı Anahtar Gruplama (PKG) gibi yük dengeleme yaklaşımları incelenmiştir. Mevcut yaklaşımlar ile önerilen yeni yaklaşım, Dinamik Anahtar Gruplama (DKG), detaylıca aktarılmış ve performansları farklı veri kümeleri ile test edilerek karşılaştırılmıştır. Her yaklaşımın uygulanabileceği ve uygulanamayacağı veri kümeleri saptanmıştır. Önerilen yaklaşımın özellikle çarpık veri kümelerinde daha başarılı olduğu gözlemlenmiştir.

Kesim 2'de Akan-veri Katarı İşleme Sistemleri hakkında bilgilere yer verilirken, bu sistemlerden çalışmamızda kullandığımız Storm ve çalışma mimarisi detaylı olarak aktarılmıştır. Kesim 3'te mevcutta bulunan yük dengeleme yöntemleri tartışılmış olup, Storm kütüphanesinde hali hazırda gelen Anahtar Gruplama (KG) ve Karışık Gruplama (SG) gibi yöntemler incelenmiştir. Bu yöntemlerin yetersiz kaldığı durumlar için önerilen yeni yöntemler de bu kapsamda incelenmiş ve detayları aktarılmıştır. Kesim 4'te ise çarpık veri kümelerinde performanslı olmayan bu yöntemlere karşı yeni bir yöntem önerilmiş ve Dinamik Anahtar Gruplama (DKG) olarak adlandırılmıştır. Önerilen bu yöntemin çalışma prensipleri ve yöntemin detayları aktarılmıştır. Kesim 5'te önerilen yaklaşımın test edilebilmesi için kullanılan kaynaklar belirtilmiş olup, ortamların kurulması ve ilgili yazılım bileşenleri hakkında bilgi verilmiştir. Kesim 6'da uygulamanın çalıştırılması sonrası alınan sonuçları değerlendirilmiştir. Bu kapsamda karşılaşılan problemlere de yer verilmiştir. Kesim 7'de sonuçlar üzerinden çıkarsamalar yapılmış, yöntem tartışılmış ve gelecek çalışmalardan bahsedilmiştir.

2. AKAN VERİ İŞLEME YÖNTEMLERİ

2.1. Veri İşleme Yapıları

Veri işleme yöntemleri yapısal olarak Bağımsız ve Dağıtık olarak ikiye, verilerin işlenişi bakımından ise Toplu ve Akan-veri İşleme olarak ikiye ayrılmaktadırlar.



Şekil 2. Veri İşleme Uygulamaları

Şekil 2'de, belirtilen yöntemlerin dağılımı ve uygulamaları görülmektedir.

2.1.1. Bağımsız

Bağımsız yapılarda bilgisayar kümeleri bulunmamaktadır. Yani sistem tek bir fiziksel makine üzerinde çalışmaktadır. Mimari yatay büyümeyi desteklememektedir ve sadece dikey büyüme ile CPU, RAM gibi kaynakların artırımı mümkün olmaktadır. İşlemler tek bir makine üzerinde yapıldığı için uygulama bütün verilere hakimdir ve başka kaynaklardan bilgi alma ihtiyacı yoktur.

2.1.2. Dağıtık

Dağıtık yapılarda ise bilgisayar kümeleri bulunmaktadır. Yani sistem birden fazla fiziksel makine üzerinde eşzamanlı çalışabilmektedir. Mimari yatay olarak büyümeye elverişlidir ve çalışacak bütün uygulamalar birden fazla makinenin çalıştığını öngörerek işlemlerini planlamalıdır. Veriler birden fazla makineye dağıtıldığı için, makineler arası

koordinasyonun yönetilmesi ve anlamlı sonuçların üretilebilmesi için diğer makinelerdeki bilgilerin de alınarak tek bir noktada birleştirilmesi sağlanmalıdır.

Moore yasası [2] her 18 ayda bir tümleşik devre üzerine yerleştirilebilecek bileşen sayısının iki katına çıkacağını, bunun bilgisayarların işlem kapasitelerinde büyük artışlar yaratacağını söylemektedir. Her ne kadar teknoloji hızla ilerlese de makinelerin de fiziksel sınırları vardır. Zaman içerisinde CPU hızları, içlerine konumlandırılan geçirgeçlerin (transistörlerin) sayısı ile orantılı olarak artmıştır. Ancak Bir CPU'nun içine yerleştirilebilecek geçirgeçlerin de fiziksel bir sınırı vardır. Fiziksel sınırlara ulaştığımız zamanlarda dikey büyüme yerini yatay büyümeye bırakmıştır. Çünkü bir makine ne kadar güçlü olursa olsun, iki makine ondan daha güçlü olacaktır. Ayrıca yatay büyümenin teorik olarak bir sınırı bulunmadığından, dağıtık bir mimariye istenildiği kadar makine eklenebilmektedir.

2.1.3. Toplu Veri İşleme

Toplu veri işleme, büyük hacimli verinin zamandan bağımsız olarak bir anda işlenmesini ifade eder. Yani verinin gelişi ile işlenişi arasında ilişki yoktur. Bir süre boyunca gelen veriler bir yerde toplanır ve belirli zaman aralıklarında toplu halde işletilirler. Bu yöntemde verilerin işleneceği zaman, işlenecek verinin tamamı kontrol altındadır; yani büyüklüğü bellidir. Örneğin, bir Telekom operatörü, telefon görüşmelerine ait kayıtları aylık olarak toplayıp, bir sonraki aya girilmesiyle toplu halde işleyerek üzerinden çıkarsamalar yapabilir. Bu senaryoda veri işleme uygulaması ayda bir kez çalıştırılmaktadır ve belirli bir numaranın aranma durumu veya bir ilde yapılan toplam arama sayısı gibi istatistiki bilgiler ancak bir ay sonra elde edilebilmektedir.

2.1.4. Akan-veri İşleme

Akan-veri işleme, toplu veri işlemenin aksine, süreklilik ve devinim içerir. Veriler toplanarak bir anda işlenmek yerine, akan-veri haline getirilerek birer birer işlenir. Akan-veri işleme yöntemleri, olay-eksenli sürekli işleme yapma yeteneğine sahiptir. Bununla birlikte veriyi işleyip sonuç üretmek için herhangi bir süre kısıtları yoktur.

2.1.5. Gerçek Zamanlı Veri İşleme

Gerçek zamanlı veri işleme ile akan-veri işleme çok yakın anlamlara sahip olduklarından genelde karıştırılabilir. Bu nedenle ayrımını iyi kavramak gerekir. Gerçek zamanlı veri işleme uygulamaları sürekli çalışmak durumundadır; yani sisteme sürekli veri girişi ve çıkışı yapılır. Ayrıca, akan-veri işleme yöntemine kıyasla zaman kısıtı vardır. Yani sisteme giren veri en kısa sürede işlenerek çıktı üretilmelidir. Evimizdeki televizyonları buna örnek gösterebiliriz. Televizyonlar uydudan sürekli akan görüntü verisini en kısa sürede işleyerek ekrana vermek durumundadırlar. Aksi takdirde bir canlı yayını dahi birkaç dakika gecikmeli olarak izleyebiliriz.

Gerçek zamanlı veri işleme, akan-veri işleme yöntemlerinin zaman kısıtı ile sürekli çalışmaları ile oluşturulur. Yani akan-veri işleme algoritmaları ne kadar verimli ve hızlı çalışırsa biz de o kadar gerçek zamanlı sonuçlar elde etmiş oluruz. Bazı verilerin işlenmesi daha fazla süre alabilir. Veri türüne ve çalışma içeriğine göre kabul edilebilen bu durumlarda ise yaklaşık gerçek zamanlı sonuçlar elde etmiş oluruz.

2.1.6. Hibrit Veri İşleme

Lambda mimari olarak da anılmakta olan bu yapı büyük hacimli verileri işlerken hem toplu veri işleme hem de akan-veri işleme metotlarının birlikte kullanımını sağlamaktadır. Bu sayede toplu veri işleme ile gecikme süresi, üretkenlik, hata dayanıklılığı konularında verimlilik sağlarken, akan-veri işleme ile de değişimlerin anlık takibi sağlanır.

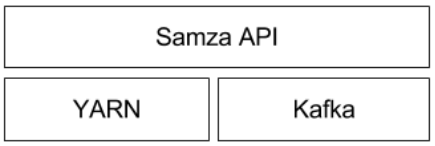
2.2. Akan-veri İşleme Araçları

Akan-veri işleme, büyük veri konusu olmanın haricinde, yeni nesil bir programlama yaklaşımı olarak da benimsenmiştir. Basit bir dizi üzerinde işlem yaparken dahi akan-veri dinamikleri kullanılmaya başlanmıştır. Bu kadar yaygınlaşan bir yapı için de birçok açık kaynak kodlu akan-veri işleme aracı geliştirilmiştir. Genel olarak hepsinin çıkış amacı ve çözdüğü problem birbirine yakındır ancak zaman içindeki gelişmeler ve çözüm yöntemleri itibariyle farklılık gösterebilmekte ve tercih edilebilmektedir. Akan-veri işleme araçlarından bazıları aşağıda verilmiştir.

2.2.1. Samza

Apache Samza [3], dağıtık tabanlı bir akan-veri işleme (AVİ) kütüphanesidir. Yapısı gereği Kafka'ya sıkıca bağlıdır. Her ne kadar birçok AVİ uygulaması Kafka'yı kullanıyor olsa da Samza, Kafka'nın mimarisinin ve verdiği altyapısal garantilerinden faydalanmak üzere özel olarak tasarlanmıştır. Kafka'nın mesajlaşma altyapısı ile YARN'ın hata toleransı, işlemci yalıtımı ve kaynak yönetimi özelliklerini kullanır. Şekilde de görüleceği üzere, Samza üç katmandan oluşmaktadır:

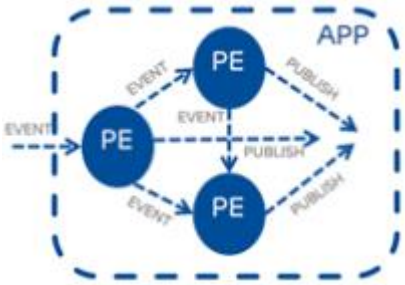
1. Akan-veri Depolama: Kafka
2. Çalıştırma: YARN
3. Veri İşleme: Samza API



Şekil 3. Samza Mimarisi [3]

2.2.2. S4

Apache S4 [4] bir diğer AVİ kütüphanesidir. Dağıtık, ölçeklenebilir, hata toleranslı ve yüksek performanslı çalışabilmektedir. Veri işleme karmaşıklığını soyutlamak ve son kullanıcıdan saklamak amaçlanmıştır. Şekil 4'te görüleceği üzere S4 mimarisi processing element (PE) adı verilen işlem birimlerinin üzerine kurgulanmıştır. Bütün işlem birimleri birbirinden bağımsız şekilde çalışmakta ve birimler arası iletişim verilerden oluşan mesajlar ile sağlanmaktadır.



Şekil 4. S4 Mimarisi [4]

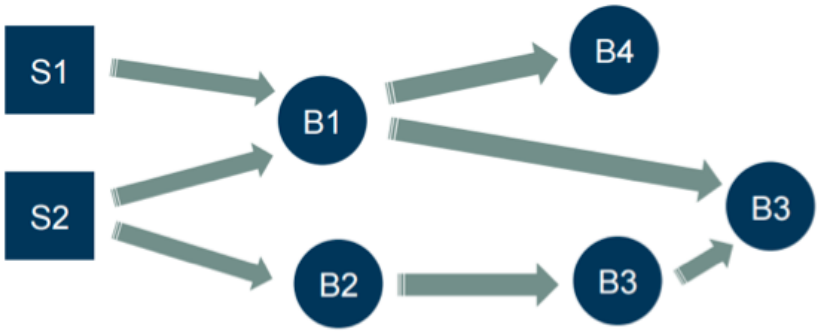
2.2.3. Spark

Spark [5], temelde toplu veri işleme üzerine özelleşmiş bir platform olmasına karşın, küçük-veri-kümeleri (micro-batching) kullanarak akan-veri işleme mimarisini de desteklemektedir. RDD (Resilient Distributed Datasets – Elastik Dağıtık Veri Kümeleri) denilen veri modelini kullanarak işlemleri tamamen bellek üzerinde yapıyor olması ile yüksek performans sağlarken, directed acyclic graph (DAG) yapısının kullanılması ile de sistemdeki bütün bileşenlerin birbirleriyle koordineli ve uyumlu çalışması sağlanmaktadır. Çok iyi geliştirilmiş bir ekosisteme sahip olmakla birlikte birçok programlama dili ile bütünleşik çalışabilmektedir.

Şekil 5. Spark Streaming [5]

2.2.4. Storm

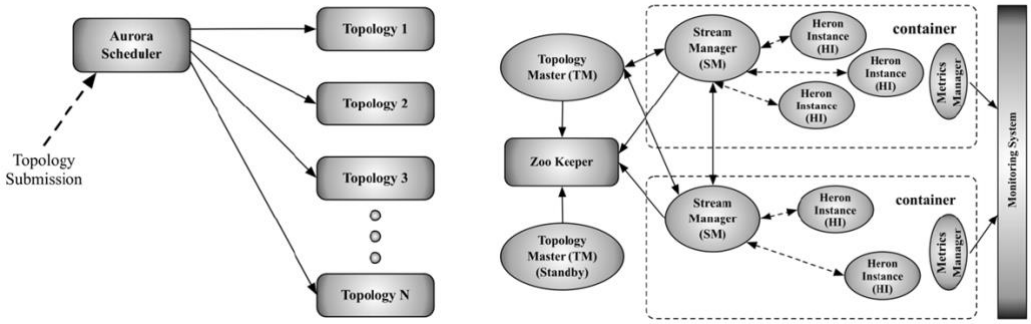
Storm [6], büyük hacimli verileri işlerken gecikme süresini en aza indirgemeye odaklanarak gerçek zamanlı çıktı üretebilen bir kütüphanedir ve yapısı itibariyle diğer yöntemlere nazaran daha düşük gecikme değerleri sağlayabilmektedir [13]. Flink ve Spark'a kıyasla daha karmaşık bir arayüze sahiptir. Bu yapıda veri kaynakları ile veriyi işleyen birimlerden oluşan ilingeler kullanılmaktadır. İlingeler akan-verilerin anlık işlenmesine odaklanmıştır.



Şekil 6. Storm İlingesi [6]

2.2.5. Heron

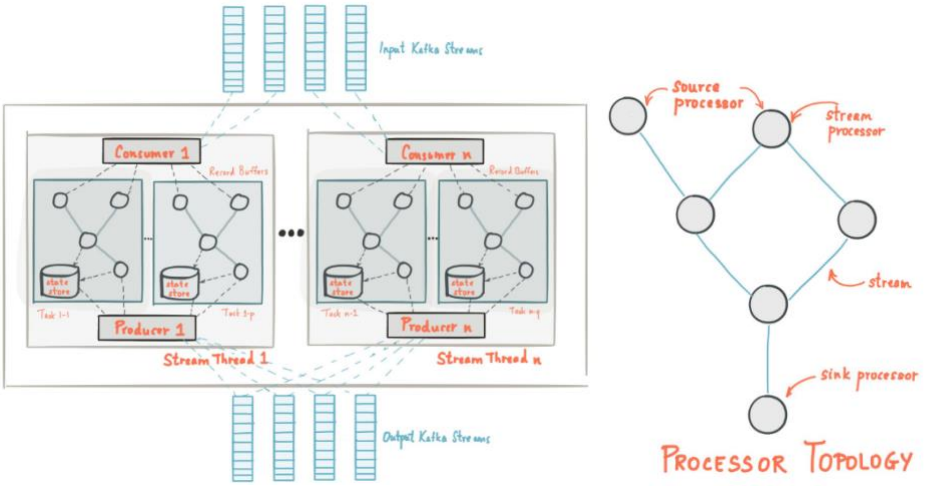
Heron [7], Storm'un varisi olarak ortaya çıkmıştır. Storm arayüzünü destekleyerek, Storm ile yazılmış uygulamaların kolayca adapte olmasını sağlarken, Storm'a kıyasla daha çok işlevsellik eklenmiş ve performans iyileştirimleri sağlanmıştır. Henüz geliştirim aşamasında olmasına rağmen iddialı performans sonuçlarına sahiptir.



Şekil 7. Heron Mimarisi [7]

2.2.6. Kafka Streams

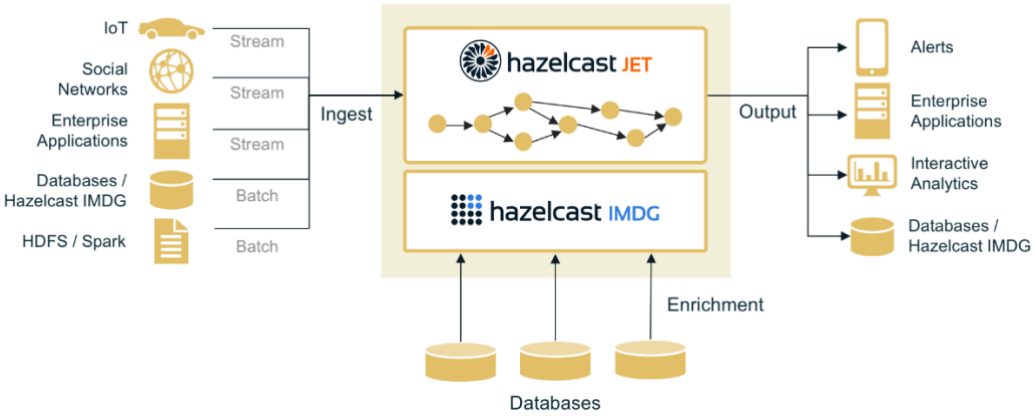
Kafka [8] veriler için bir nevi bekleme alanı oluşturan kuyruk yapısı olarak tanımlanabilir. Kafka Streams [9] ise Kafka'da tutulan verileri işleyerek ve analiz ederek çıktılar üretmeye yarayan ve bu çıktıları tekrar Kafka'ya veya başka bir dış sisteme yazan, dağıtık tabanlı AVİ kütüphanesidir. Gerek basitliği ve gerekse güçlü olması nedeniyle tercih edilmektedir.



Şekil 8. Kafka Streams İlingesi [9]

2.2.7. Hazelcast Jet

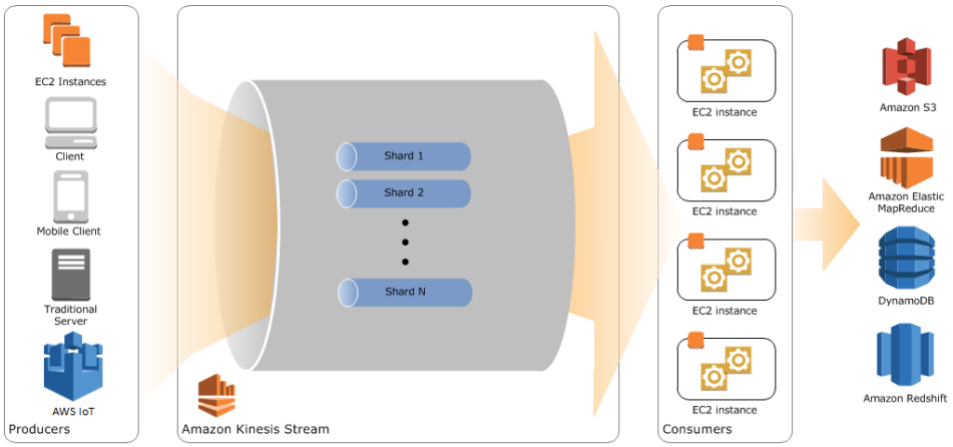
Hazelcast [10] dağıtık ortamda paylaşılan bellek kullanımı ile ortaya çıkmış başarılı bir girişimdir. Hazelcast Jet ise bu altyapıyı kullanarak büyük veri kümelerini hızlı işlemeye yarayan dağıtık veri işleme platformudur. Henüz çok yeni olmasına rağmen MapReduce, Spark, Flink gibi benzer platformlara göre daha performanslı çalışmakta ve yapısı gereği gelecek vadetmektedir.



Şekil 9. Hazelcast Jet Mimarisi [10]

2.2.8. Kinesis

Kinesis [11], Amazon Web Servisleri kapsamında verilen bir AVİ hizmetidir. Büyük hacimli verileri sürekli olarak toplayarak gerçek zamanlı veri işleme ve çıktı üretme yeteneklerine sahiptir.



Şekil 10. AWS Kinesis Mimarisi [11]

2.3. Storm Çalışma Mimarisi

Storm [6] mimarisi yapı itibariyle Hadoop mimarisine çok benzer. Hadoop'da *MapReduce* işleri koşulurken Storm'da ilingeler çalıştırılır. Storm aynı zamanda Spark ile karşılaştırılmaktadır. Aralarındaki temel fark ise Storm'da, Samza'da olduğu gibi, her olay verisi bireysel işlenirken, Spark'ta küçük-veri-kümeleri (micro-batching) olarak tabir edilen küçük parçalar halinde biriktirilen veri kümelerinin birlikte işlenmesidir [14].

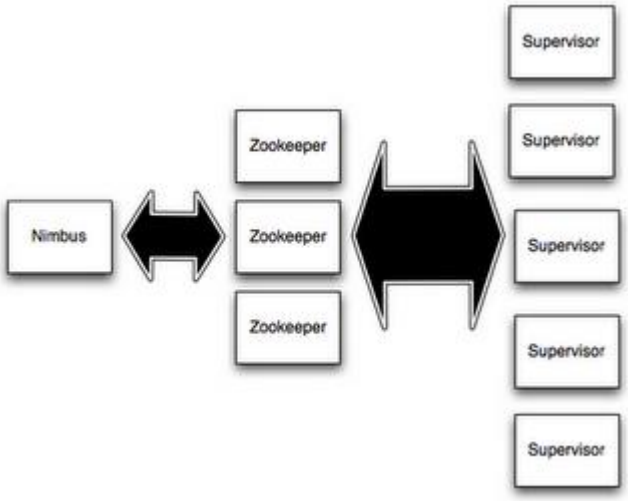
Storm yapısı itibariyle çok-öğeli akan-veri üzerine kuruludur ve belirtildiği gibi, kaynak ve işçi birimlerden oluşan yönlü çizgedir. Bu çizgeye ilinge adı verilir. İlingeler gelen veri kümesinin kaynaktan alınması, çeşitli işlemlerden geçirilmesi ve çıktı üretilmesi adımlarını içeren bir süreç yapısıdır.

2.3.1. Kümeleme (Clustering)

Storm mimarisinde iki farklı birim bulunur: Usta Birim ve İşçi Birim. Usta Birim, Hadoop mimarisindeki İş Takipçisine benzer şekilde Nimbus adında bir program çalıştırır. Nimbus çalışacak kodun kümedeki bütün makinelere dağıtılmasından, makinelere görevlerin atanmasından ve makinelerde oluşacak hataların yakalanmasından sorumludur. Her İşçi Birim ise Yönetici Birim adında bir program çalıştırır. Yönetici Birim ise kendisine atanacak görevleri dinleyerek, Nimbus'tan gelecek yönergelere göre iş süreçlerini başlatmaktan ve sonlandırmaktan sorumludur. İlinge, kümedeki bütün makinelere dağıtılmış şekilde birçok İşçi Birim bulundurur ve her İşçi Birim ilingenin bir alt kümesini çalıştırmaktadır.

Nimbus ve Yönetici Birim programları durumsuz olarak çalışmaktadır ve hata oluşması durumunda işlemi hemen sonlandırmaktadır. Yani, çalışan makinelerden birinin iş yapamaz duruma gelmesi halinde, kalan makineler işleri devralıp çalışmaya sorunsuzca devam edebilmektedirler. Makinelerin durumsuz çalışabilmeleri, sistemin hatalara karşı dayanıklı olmasını ve daha stabil olmasını sağlamaktadır. Makineler durum bilgisi tutmadıklarından, Nimbus ve Yönetici Birim arasındaki iletişimi ve koordinasyonu sağlayan başka bir yapı bulunması gerekmektedir. İşte bu koordinasyonun sağlanmasından Zookeeper [12] sorumludur. Zookeeper Nimbus ve Yönetici Birim arasında bir soyutlama yapmakta; bütün durum bilgilerini kendi yerel

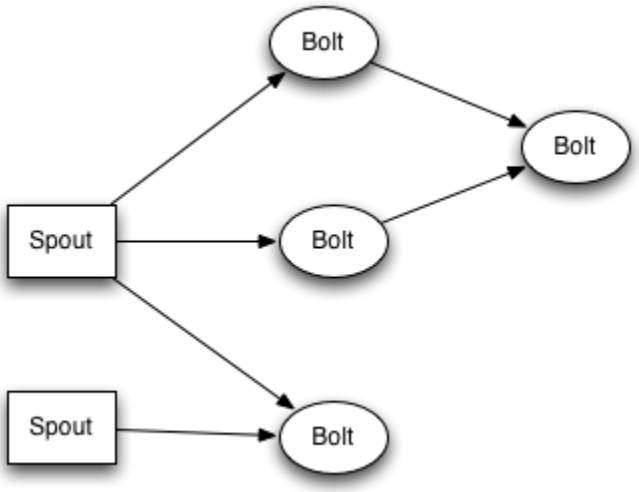
diski üzerinde tutarak ve gerekli yönlendirmeleri yaparak Nimbus ile Yönetici Birim arasındaki iletişimi ve koordinasyonu sağlamaktadır.



Şekil 11. Storm Kümeleme Yapısı [6]

2.3.2. İlinge (Topology)

İlinge, gerçek zamanlı çalışan bir uygulama mimarisinin Storm yapısı üzerindeki gerçekleştirimidir. Bir ilinge, Kaynak Birim ve İşçi Birimlerden oluşan yönlü bir çizgedir ve bu yapıda birimler arası veri iletişimi gruplama yöntemlerine göre yapılır. Yapı olarak MapReduce mimarisine eş olarak düşünülebilir ancak MapReduce mimarisinde veri işleme eninde sonunda tamamlanırken, Storm ilingesi sonsuza kadar çalışacaktır.



Şekil 12. Storm İlingesi [6]

2.3.3. Çok-öğeli (Tuple)

Storm mimarisinde veri dönüştürülerek iletilir ve bu iletim çok-öğeli aracılığıyla gerçekleştirilir. Çok-öğeli, Kaynak Birim-İşçi Birim veya İşçi Birim-İşçi Birim arasında iletilebilecek en küçük veri paketi olarak tanımlanabilir. Çok-öğeli, temel olarak, integer, long, short, byte, string, double, float, boolean, byte array türlerini içerebileceği gibi kullanıcının kendi serileştirme yapısını kurarak istediği türden veriyi içermesine de olanak sağlamaktadır.

2.3.4. Akan-veri (Stream)

Akan-veri, Storm mimarisinin temel soyutlamalarından biridir. Sınırsız sayıdaki çok-öğeli serisini ifade eder. Akan-veriler, çok-öğeli içindeki verileri gösteren bir şema ile birlikte tanımlanırlar.

2.3.5. Kaynak Birim (Spout)

Kaynak Birim, bir ilingedeki akan-verinin kaynağıdır. Kaynak Birimler, ilingeye girecek akan-verinin giriş noktasıdır ve genellikle dış kaynaklardan çok-öğelileri okuyarak ilingeye yollarlar. Bu kaynaklar bir kuyruk, bir API veya herhangi bir veri kaynağı olabilir. Kaynağın görevi veri kaynağıyla iletişim kurarak veriyi sürekli olarak almak, bu verileri çok-öğelilere çevirmek ve son olarak ise işlenmek üzere işçilere aktarmaktır.

2.3.6. İşçi Birim (Bolt)

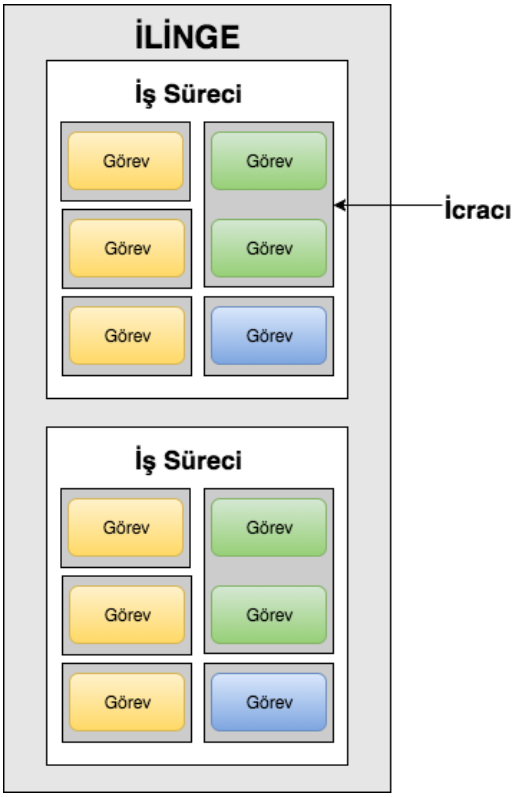
İşçi Birimler, girdi olarak akan-verileri alarak üzerinde birtakım işlemler uygularlar ve çıktı olarak yeni akan-veriler üretirler. Karmaşık veri dönüşüm algoritmaları birçok adımdan oluşacağı için doğal olarak bu işlemi yapacak ilinge de birçok İşçi Birimden oluşmak durumundadır. İşçi Birimler iş yapan birimlerdir ve fonksiyon çağırmak, çok-öğelileri filtrelemek, akan-verileri bütünleştirmek, toplamak, veritabanlarıyla konuşmak gibi birçok görevi yapabilirler. İşçi Birimlerin çıktıları bir diğer İşçi Birime girdi üretebileceği gibi, çıktı üretmeyip akışı da sonlandırabilir.

2.3.7. İş Süreci, İcracı, Görev (Process, Executor and Task)

Storm kümesi içerisindeki bir makine, bir veya birden fazla ilinge için, bir veya birden fazla İş Süreci çalıştırabilir. İş Süreci ilingenin bir alt kümesini çalıştırır ve bunu kendi üzerindeki Java Sanal Makinesinde (JVM) gerçekleştirir. Her İş Süreci belirli bir ilingeye aittir ve ilingenin bileşenleri (Kaynak Birim veya İşçi Birim) için bir veya birden fazla İcracı çalıştırabilir.

İcracı, İş Sürecinin içinde oluşturulan ve onun Java Sanal Makinesi üzerinde çalışan bir iş parçasıdır. Bir İcracı, aynı bileşen (Kaynak Birim veya İşçi Birim) için bir veya birden fazla Görev çalıştırabilir. Bir İcracının her zaman sadece bir iş parçası vardır ve bunu bütün Görevler için ortak kullanır. Yani Görevler İcracı üzerinde seri olarak çalıştırılmaktadır.

Görev, asıl işi yapan birimdir ve içinde bulunduğu İcracının iş parçası içerisinde yaşar. Her Kaynak Birim ve İşçi Birim ilingeye ait makineler üzerinde birçok Görev çalıştırabilmektedir ve gruplama tercihi verinin Görevler arasında nasıl dağıtılacağını belirlemektedir. Şekil 13'te, bir ilingedeki paralel çalışma yapısı gösterilmektedir [15].



Şekil 13. Storm İlingesi [15]

Bir ilinge içerisinde birden fazla İş Süreci tanımlanabilmektedir. Tanımlanan İcracılar ve Görevler, İş Süreçleri arasında dengeli olarak bölüştürülürler.

2.3.8. Gruplama

Storm mimarisinde İşçi Birimler arasında yönlendirme belirlenen yöntemler doğrultusunda yapılmaktadır. Bu yönlendirmeler aynı zamanda ilingenin verimini doğrudan etkiler; çünkü bu yönlendirme algoritmaları iş yükünün makineler arasında dağıtımından sorumludur. Bu da demek oluyor ki bu algoritmalar ne kadar verimli olursa yük o kadar dengeli dağıtılmış ve bütün makineler verimli kullanılmış olacaktır. Bu kapsamda yönlendirme algoritmalarını yük dengeleme yöntemleri olarak da adlandırabiliriz.

Storm kütüphanesinde tanımlı gelen birçok gruplama yöntemi mevcuttur. Bunlardan en bilinen iki temel yük dağıtım yöntemi Karışık Gruplama ve Anahtar Gruplamadır. Tanımlı gelen bu algoritmalar bu bölümde incelenecek olup, bu algoritmaların yetersiz

veya verimsiz kaldığı durumlar için geliştirilen yeni yaklaşımlara ise Kesim 3'te yer verilecektir.

2.3.8.1. Karışık Gruplama (Shuffle Grouping)

Çok-öğeliler, İşçi Birimlere rastgele dağıtılırlar. Bu yapıda her İşçi Birimin eşit yük aldığı garanti edilir. Aynı alan değerine sahip çok-öğeliler her İşçi Birime gönderilebilir.

2.3.8.2. Anahtar Gruplama (Key/Field Grouping)

Akan-veri, çok-öğeli içerisinde tanımlanan alanlara göre ayrıştırılır ve İşçi Birimlere yönlendirilir. Bu yapıda aynı anahtar değerine sahip çok-öğelilerin aynı İşçi Birime gönderileceği garanti edilir. Örneğin, kullanıcı adına göre bir gruplama yapıldığını düşünürsek, aynı kullanıcı adına sahip çok-öğeliler her zaman aynı İşçi Birime gönderilecektir.

2.3.8.3. Bütün Gruplama (All Grouping)

Akan-veri, bütün işçilere kopyalanarak gönderilir. Yani, Rastgele ve Anahtar Gruplamanın aksine, gelen çok-öğeli, tek bir İşçi Birim yerine bütün İşçi Birimlere aynı anda gönderilir.

2.3.8.4. Küresel Gruplama (Global Grouping)

Bütün akan-veriler, sadece tek bir İşçi Birime gönderilir. Sistemdeki en küçük kimlik bilgisine sahip olan Görev hedef olarak seçilir.

2.3.8.5. Hiç Gruplama (None Grouping)

Bu yöntem, gruplamanın önemli olmadığı durumlarda kullanılır. Şu anda Karışık Gruplama (SG) ile aynı şekilde çalışmaktadır.

2.3.8.6. Doğrudan Gruplama (Direct Grouping)

Özel bir gruplama türüdür. Bu yapıda hedef İşçi Birim doğrudan belirlenmiştir ve çıktıyı üreten İşçi Birimi, çok-öğelinin hangi İşçi Birime gönderileceğini bilmektedir.

2.3.8.7. Yerel veya Karışık Gruplama (Local or Shuffle Grouping)

Hedef İşçi Birim, aynı İş Süreci içerisinde bir veya daha fazla Görev çalıştırıyorsa, çok-öğeli bunlar arasında rastgele dağıtılacaktır. Aksi taktirde Karışık Gruplama yapacaktır.

2.3.8.8. Özelleştirilmiş Gruplama (Custom Grouping)

Çok-öğelilerin İşçi Birimlere nasıl dağıtılacağı programatik olarak tamamen kontrol edilebilmektedir. Kullanıcının tanımlayacağı algoritma dağıtımın nasıl yapılacağını belirlemektedir. Bu kapsamda kullanıcı, verinin içeriğine ve/veya üstverisine bakarak hedef İşçi Birim belirleyebilmektedir.

3. YÜK DENGELEME YÖNTEMLERİ

Akan-veri işlemede, sahip olduğumuz makineler arasında yükü dengeli dağıtmak, sistemimizin verimini ve üretkenliğini arttırırken, çalışma süresini azaltarak daha kısa zamanda daha hızlı sonuçlar elde etmemizi sağlamaktadır. Bu kapsamda, gelen veriyi en kısa zamanda işleyebilmek ve yüksek üretkenlik elde edebilmek için, yükü mümkün olduğunca eşit dağıtabilmemiz gerekmektedir.

3.1. Yük Dağıtımı ve İşlenen Veri İlişkisi

Yük dağıtımı, doğrudan verinin içeriğine bağlıdır. Bir verinin diğer verilerle doğrudan ilişkisi yok ise, yani durumsuz ise, veri içeriğine bakılmaksızın Karışık Gruplama (SG) yapılarak İşçi Birimlere dağıtılabilir. Bu sayede, yük bütün İşçi Birimlere eşit olarak dağıtıldığından sistem maksimum performansta çalışabilir. Diğer taraftan bir verinin diğer veri ile ilişkisi var ise, yani durumlu ise, ilişkili olan verilerin bir araya getirilip toplanarak tek bir sonuç elde edilebilmesi gerekmektedir. Bu durumda ise Anahtar Gruplama (KG) yöntemi kullanılmalıdır. Bu sayede aynı anahtar değerine sahip olan verilerin aynı İşçi Birime gönderilmesi sağlanabilir ve bu İşçi Birim aynı anahtar değerine sahip bütün verilere sahip olduğundan, gerekli işlemleri yaptıktan sonra tek bir sonuç üretebilecek durumda olacaktır. Ancak bu senaryoda yükün bütün İşçi Birimlere eşit dağıtılacağının garantisi bulunmamaktadır. Yükün dağıtımı doğrudan anahtar değerine yani verinin kendisine bağlı olduğundan, yük dağıtımı gelecek veriye bağlı olarak değişkenlik gösterecektir. Örneğin, bir anahtar değeri diğerlerine kıyasla daha sık gelirse, bu verinin gideceği İşçi Birim üzerinde, gelen verinin yoğunluğuyla doğru orantılı olarak bir yük meydana gelecektir. Bu da bir makinenin diğer makinelerden daha fazla yüke sahip olacağı anlamına gelmektedir.

Orantısız veya çarpık gelen verinin sistemin performansını doğrudan etkileyebiliyor olması en son istenecek bir durumdur. Çünkü belirli zamanlarda gelebilecek aynı tür veriler sistemin performansını olumsuz etkileyebilmekte ve üretkenliğin azalarak çıktı alma sürelerinin uzamasına sebebiyet verebilmektedir. Gerçek zamanlı çalışan ve çıktı üreten sistemlerde bu tür gecikmeler pahalıya mal olabilmektedir. Ayrıca gelecek veriyi

her zaman tahmin edemeyeceğimiz için, sistemimiz stabil ve ölçeklenebilir olmayacaktır. Bu da sistemin öngörülemez bir şekilde çalışacağı anlamına gelmektedir.

Bir diğer problem ise sistemin tamamına hâkim olamıyor olmamızdır. Eğer sistemdeki bütün dağılımı tek bir noktadan yönetebiliyor olsaydık, bütün sisteme hâkim olacağımızdan yükü en iyi şekilde dağıtabilmemiz de mümkün olacaktı. Ancak dağıtık mimariye sahip olan sistemlerde, her şeyi bilen tek bir makineye sahip olmak çok pratik olmamakla birlikte önerilmemektedir [26]. Bu nedenle, dağıtık sistemlerde yük dengeleme sorununu çözmek için alternatif çözümler üzerinde yoğunlaşmamız kaçınılmaz olmaktadır.

3.2. İlgili Çalışmalar

Özellikle bilim ve endüstri alanındaki gelişmelerle birlikte büyük veri uygulamalarına olan ihtiyacın artmasıyla, büyük miktarlardaki akan-veriyi veya olay verilerini çözümleyebilen ve işleyebilen çeşitli çözümler önerilmiştir [27, 28].

Akan-veri işleme (AVİ) uygulamaları, büyük miktarda verinin işlenerek gerçek zamanlı sonuçlar elde edilmesi ihtiyacıyla birlikte daha da çok önem kazanmıştır. [29], ölçeklenebilir AVİ uygulamalarında kaynak tüketimini en düşük seviyeye indirirken, gecikme sürelerini kontrol altında tutabilen tepkisel bir çözüm sunmaktadır. [30], varsayılan bölüntüleme karmasıyla (hash partitioning) karşılaştırıldığında, isteğe özel uyarlanabilen bölüntüleme yöntemlerinin, bütünleştirilecek sonuç durumlarının boyutunu azaltarak bellek alanından tasarruf ettiğini göstermektedir. [31], yüksek bağlılaşım (correlation) içeren veriler için bağlılaşım duyarlı çok yönlü bir akış sorgu eniyileyici (CMR, correlation-aware multi-route stream query optimizer) kullanan etkili bir bölüntüleme yöntemi önermektedir. Bu yöntem, benzer istatistiksel özelliklere sahip olan çok-öğelilerin aynı süreci takip ederek en iyi şekilde işleneceği düşüncesine dayanan çok yönlü bir eniyileyici yaklaşımı sergilemektedir [26]. Çok yönlü eniyileyici, veri katarını önce birçok bölüme ayırır ve her bir bölüm birleşimi için ayrı bir sorgu planı oluşturmaktadır. [32], koşut akan-veri işleme için uyarlanabilir bir girdi kabulü ve yönetim yöntemi önerirken, veri akış davranışları ve sorgu işleme katmanı gereksinimleri hakkındaki mevcut bilgileri alarak, veri akışının sisteme nasıl giriş yapacağına kendiliğinden karar vermektedir. Öte yandan, ölçeklenebilir güçlü bir

sistem oluşturmak, yüksek hacimli verileri çok kısa sürede işlemek durumunda olan, Heron [33] gibi bir AVİ yöntemi kullanan Twitter gibi uygulamalar için oldukça önem arz etmektedir.

Büyük hacimli verilerin işlenerek nihai sonucun hızlı üretilmesi gerektiğinde, verinin içeriği de önem kazanmaktadır. Günümüzde, dağıtık AVİ motorlarında Karışık Gruplama (SG) yöntemlerinin oldukça çalışılmasına rağmen [34], durumlu veri kümeleri için önerilen gruplama yöntemleri her zamankinden daha çok önem kazanmıştır. [35] dinamik ölçeklenebilir bir yapı ve durumlu işleçlerin iyileştirilmesi için bütünleşik bir yaklaşım tanımlarken, [36] LinkedIn'de kullanılan, durumlu ölçeklenebilir bir AVİ çözümü olan Samza'yı tanıtmaktadır. Bu çalışma aynı zamanda dinamik ölçeklendirme tekniklerini de göz önüne almaktadır. Dinamik ölçekleme teknikleri, beklenmedik yük artışlarında sistemin tepkisel olarak yük artışına cevap verebilmesini ve bu sayede sistemin sürekli yüksek kapasitede kaynak kullanımının önüne geçmesini sağlamaktadır [37, 38, 39].

Dinamik yük dengeleme yöntemleri de son zamanlarda oldukça önem kazanmıştır ve bu yöntemler büyük hacimli verileri ölçeklenebilir bir yapıda işleyebilmek için bilgisayar kümeleri kullanmaktadır. Dağıtık akan-veri işleme yöntemleri, özellikle Facebook ve Twitter gibi, büyük hacimli akan-verileri ölçeklenebilir bir yapıda işleme gereksinimi duyan yazılım şirketleri tarafından yaygın bir şekilde benimsenmiştir. Bununla birlikte, dağıtık veri işleme sistemlerinin bileşenleri arasında yükü dinamik olarak dengeli dağıtmak, yüksek veri hacmi, bileşenlerin durum bilgilerinin yönetim ihtiyaçları ve düşük gecikmeli işleme gereksinimleri nedeniyle oldukça zor olabilmektedir. Bu nedenle, [40] gerçek zamanlı olarak Karmaşık Olay İşleme (CEP – Complex Event Processing) motorlarının yükünü dinamik olarak dengeleyen ve iki dengeleme yöntemini kullanarak veri akışı hacmindeki ani değişikliklere uyum sağlayan bir çözüm getirmektedir. [41], küresel bilgi gerektirmeyen, yerel bir yük dengeleme yöntemi tanıtırken, [42], S4 koşut akan-veri işleme motorları için dinamik bir yük dengeleme yöntemi tanıtmaktadır. [43] ise kaynak çeşitliliği ve paylaşımı, iletişim gecikmesi, bir makineden diğerine iş göçü ve yük dengelemesi gibi bilgisayar şebekeleri için zorlu olan birkaç konuyu ele almaktadır.

Büyük hacimli verileri işlerken iyi ve dinamik bir yük dengelemesinin başarılı olabilmesi için, sistemin, verilerin içeriğini de dikkate alarak, kümedeki makinelere trafiği eşit olarak dağıtabilmesi gerekmektedir. Karışık Gruplama (SG) yönteminin aksine, bu tür algoritmalar genellikle Anahtar Gruplama (KG) veya Akan-veri bölüntüleme olarak adlandırılırlar. Bu yöntemler, genel olarak verileri dinamik olarak bölüntüleyerek dağıtırlar. Anahtar bölüntüleme yöntemleri bağlamında, verimli ve en iyiye yakın yük dengelemeyi başarabilmek için gerek MapReduce ortamında [44, 45, 46], gerekse de AVİ sistemleri üzerinde çalışılmıştır [47, 48, 49].

Yük dengeleme problemi, aynı zamanda verinin içerik temelli yönlendirme problemi (content-based routing) olarak da incelenmektedir [50, 51, 52, 53, 54, 55]. İçerik temelli yönlendirme, akıllı yönlendirme sunucuları tarafından verilere basit yönlendirme kuralları uygulayarak gerçekleştirilebilmektedir. Akıllı içerik temelli yönlendirme teknikleri, verimli ve uyarlanabilir yönlendirme sağlayarak, uçtan uca gecikme ve üretkenlik performansını iyileştirmek için önerilmiştir.

Nihai sonucu elde etmek için ek bir işlem gerektiğinden, verinin çarpık olması dağıtık mimarilerde problem oluşturabilmektedir. Bu kapsamda, verinin çarpık olması durumunda yük dengeleme problemi de incelenmiştir [56, 57, 58, 59]. [60], durum temelli işleçler için dinamik iş yükü atamasını destekleyecek pratik algoritmalara sahip yeni bir anahtar tabanlı iş yükü bölüntüleme çözümü sunmaktadır. [61] ise, her İşleme Öğesinin (PEI – Processing Element Instance), mevcut iş yükü kapasitesine göre işleyişini sağlayan, Tutarlı Gruplama (CG – Consistent Grouping) adlı yeni bir bölüntüleme yaklaşımı önermektedir. Bu yaklaşımdaki temel amaç, yükün makinelere mevcut iş yüklerine ve kapasitelerine göre dengeli dağıtılabilmesidir.

Öte yandan, bizim önerimize (DKG) benzer şekilde, [62] akan-veri içerisinde en sık geçen anahtarları etkin bir şekilde tanımlamak için yeni bir yük dengeleme tekniği önermektedir. Sık geçen anahtarlar, dengeli bir yük dağıtımı sağlayabilmek için $d \geq 2$ seçeneğine atanmaktadır. Burada d, işleç dağıtımının bellek ve hesaplama maliyetini en aza indirgemek için otomatik olarak ayarlanmaktadır. Yöntem çevrimiçi çalışırken, yönlendirme tablolarının kullanılmasını gerektirmez. Morales, bu zorlu problem için iki

yeni yöntem önermektedir: D-Choices ve W-Choices. Bu yöntemler, anahtar dağılımının başını oluşturan akımdaki sık geçen anahtarları izlemek için bir yöntem kullanmakta ve bu sayede sık geçen anahtarların daha geniş bir makine grubunda işlenmesine izin vermektedir.

3.3. Dağıtık Mimarilerde Yük Dengeleme

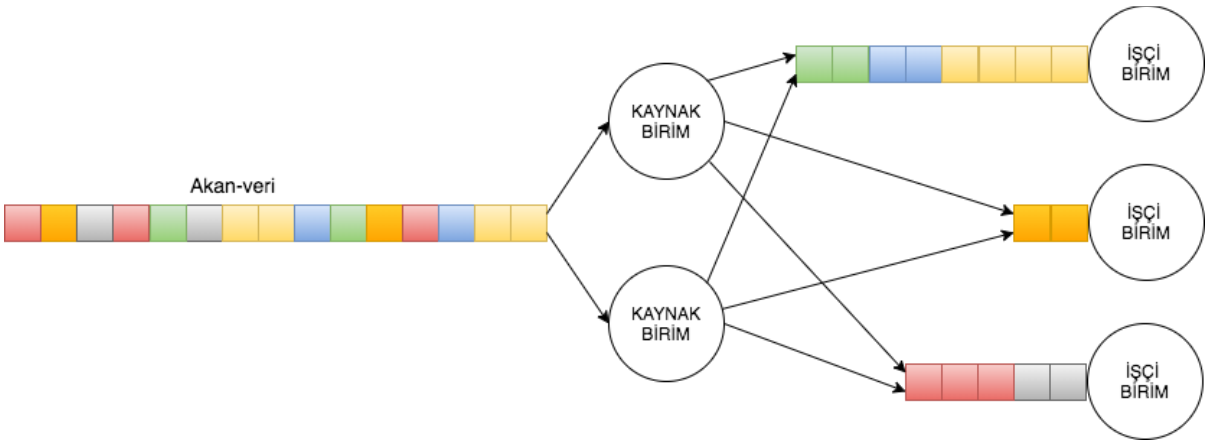
Yük dengeleme, gerçek zamanlı veri işleme ihtiyacının artmasıyla ve en kısa sürede sonuç üretilebilmesi gereksiniminin var olmasıyla daha da önemli hale gelmiştir. Bu kapsamda S4, Storm ve Samza gibi dağıtık mimariye sahip veri işleme sistemleri daha da popüler hale gelmiştir. Çünkü bu sistemler büyük hacimli verileri kümelenmiş bilgisayarlar üzerinde çok az gecikmeler ile gerçek zamanlı işleyebilmektedirler. Yük dengeleme konusunda teorik ve pratik birçok araştırma yapılmış ve birçok yaklaşım önerilmiştir. Bu kesimde yapılan bu çalışmalardan ve önerilen yaklaşımlardan bahsedilecektir.

Dağıtık mimarilerdeki yük dengeleme problemi, toplar-ve-sepetler (balls-and-bins) problemi olarak da görülmüş ve iki hedef makine seçmenin gücü teorik olarak araştırılmıştır [23, 25]. Bu problemde N tane sepetimizin olduğu ve içerisine M tane topun atıldığı varsayılmaktadır. Bütün topların sepetlere dağıtımı tamamlandıktan sonra, sepetlerin doluluk oranları karşılaştırılmaktadır. Dağıtık sistemlerin çalışmasına oldukça benzeyen bu yapıda toplar mesajlara sepetler ise sunuculara benzetilmiş ve bütün topların eşit ağırlığa (iş yüküne) sahip olduğu varsayılmıştır. Eğer topları anahtar değerler olarak düşünürsek, daha yoğun gelen anahtarları daha ağır toplar olarak ele almamız gerekecektir [63]. Bu problemde elde edilmeye çalışılan amaç ise topların dağıtıldıktan sonra ağırlığın, yani bütün topların, sepetlere dengeli dağıtılmış olmasıdır. Öte yandan, geleneksel yöntemlerde topların tek bir kaynaktan geldiği düşünülmektedir ve dağıtık sistemlerde birden fazla kaynaktan top geleceği düşünüldüğünde sepetler arası koordinasyon ve topların küresel ölçekte dağıtımı da bu problemin kapsamına girmektedir. Bu problem üzerinde de yapılmış birçok çalışma ve araştırma bulunmaktadır [64, 65, 66].

Akan-veri işleme uygulamaları DAG (Directed Acyclic Graph), yani Yönlü Döngüsüz Çizge ile ifade edilmektedirler. Bu çizgede, köşeler İş Yapan Birimi, kenarlar ise bu birimler arasındaki veri akışını ifade etmektedir.

MapReduce'de olduğu gibi, dağıtık mimariye sahip ve akan-veri işleyen motorlarında da akan-veriyi gruplayarak farklı makinelere dağıtma işi genellikle anahtar değerleri üzerinden yapılmaktadır. Anahtar Gruplama olarak da anılan bu bölüntüleme yaklaşımı ile aynı anahtara sahip mesajların aynı iş birimi tarafından ele alınacağı garanti edilebilmektedir. Hedef makine belirleme işlemi ise karma fonksiyonları kullanılarak yapılmaktadır. Karma fonksiyonları sayesinde gelen mesajlara dair bir durum bilgisi tutulmadan, sadece anahtar değerlere bakılarak hedef makine belirlenebilmektedir. Öte yandan, karma fonksiyonu her anahtar için tek bir değer ürettiğinden yükün dengeli dağıtılamaması durumu ortaya çıkabilmektedir.

Şekil 14'te, gelen verinin çarpık olması durumunda ortaya çıkan yükün dengesiz dağıtımı görselleştirilmiştir.



Şekil 14. Yükün Dengesiz Dağıtımı [26]

Yükün dengesiz dağılımı ise kaynakların verimsiz kullanımına ve sistemin çalışma ve çıktı üretme performansının düşmesine yol açmaktadır. Karışık Gruplama bölüntüleme yaklaşımı ise Round Robin yönlendirmesi ile yükü mükemmel bir şekilde dağıtabilmektedir. Ancak bu yöntem mesajları makinelere dağıtırken anahtar değerine bakmadığından genelde durumsuz işlemlerde tercih edilmektedir. Durumlu işlemlerde

kullanılması içinse ekstra bellek kullanımı ve bütünleştirme aşamasının eklenmesi gerekmektedir.

Yükün dengeli dağıtımı için genel kabul gören bir başka çözüm ise yapılacak işlemin bir başka makineye taşınmasıdır [16, 17, 18, 19, 20, 21]. Kümedeki bir makinede aşırı yük tespit edildiğinde, sistem mesajların bir kısmını işlem durumlarıyla birlikte diğer makinelere aktarır. Bu sayede sistem bir süre sonra yeniden dengeye gelir. Yük paylaşımı, dağıtık işletim sistemleri ve veri tabanları başta olmak üzere birçok alanda geniş çaplı araştırılmış ve çalışılmıştır [18, 19, 20, 21]. İlk bakışta anlaşılması ve uygulanması çok kolay gelen bu yöntemin dezavantajları da vardır. En başta, sistemin dengesizlikler için hangi sıklıkla taranması gerektiğine ve hangi sıklıkla yeniden dengeleme işleminin yapılması gerektiğine karar verilmesi gerekmektedir. Bunlar genelde uygulamaya özgü olan parametrelerdir ve diğer makinelere aktarım yapılmasıyla mevcut sistemdeki yük dengesizliği arasındaki maliyetlerin karşılaştırılması ve buna göre karar verilmesi gerekmektedir. Ayrıca aktarımın yapılabilmesi için bütün makineler arasında ek bir bağlantı kurulması gerekmektedir. Bu da dağıtık mimarilerde istenmeyen ve kabul görmeyen bir yaklaşımdır. Çünkü makinelerin yüksek erişilebilirlikte olması ve aktarımdan dolayı gelecek ek ağ trafiği yükünü kaldırabilmesi gerekecektir.

Sistemin tekrar dengeye getirilmesi aşamasında makineler arası aktarım yapıldığından, daha sonra gelecek olan mesajların da yeni makinelere yönlendirilmeleri gerekmektedir. Bunun yapılabilmesi için ise dağıtım yapacak her sistemin birçok yönlendirme tablosu tutması ve her dağıtım için anahtarların ve yönlendirilecekleri makinelerin açık olarak tutulması gerekmektedir. Bu da dağıtık çalışan mimarilerde ve birçok kaynaktan milyonlarca anahtar içeren mesajlar alan sistemlerde uygulanabilir değildir; çünkü mesaj sayısının artmasıyla sistemin kullanması gereken bellek boyutu da oldukça artmaktadır.

Makineler arası aktarım yaparak yük dengelemeye çalışan bir diğer yöntem ise **Flux**'tır. Flux, her makinenin yükünü takip ederek makineleri yük durumlarına göre sıralar [16]. Yük dengesizliğinin tespit edilmesi durumunda ise, en yüklü olan makineden en az yüklü olan makineye, en yüklü olan ikinci makineden en az yüklü olan ikinci makineye

şeklinde devam ederek makineler arası aktarım yapar ve yükü dengelemeye çalışır. Flux, hata dayanıklılığına sahip, yük değiştirme yöntemidir. Durumlu işlemler için tekrar kullanılabilir, uyarlanabilen bölüntüleme ve ayrıştırma yapısı sunmaktadır. Hem yerel makine üzerinde kısa vadeli yük dengesizliklerini hem de kümelenmiş makineler üzerindeki uzun vadeli yük dengesizliklerini tespit edip gerçek zamanlı yeniden bölüntüleyerek dengeyi sağlayabilmektedir. Flux, durumlu işlemler için gerçek zamanlı yeniden bölüntüleme yaparak yükü makineler arasında çalışma anında aktarmakta, üretkenlik ve gecikme sürelerinde ise iyileşmeler elde etmektedir.

Aurora* ve **Medusa** yine makineler arası aktarım yaparak yük dengelemeye çalışan diğer yöntemlerdendir [17]. Aurora, merkezileştirilmiş bir akan-veri işleme motoru olarak tanımlanabilir. Aurora'nın dağıtık mimaride çalışabilmesi adına geliştirilmiş ve Aurora* ile Medusa önerilmiştir. Aurora*, kümedeki bütün makinelerin tek bir yönetim altına girdiği bir ortam varsaymaktadır ve bu yönetim altında birçok tekil Aurora sunucusu barındırmaktadır. Bu kapsamda Aurora*, makinelerin içinde bir dağıtım yapısı sağlamaktadır. Öte yandan Medusa, kümedeki bütün makinelerin federe bir yapı içerisinde çalışabileceği bir altyapı sunarken makineler arası bir dağıtım yapısı sağlamaktadır. Bu kapsamda sistem, makinelerin kendi aralarında iletişim kurarak yük aktarımı yaparak ve kendiliğinden dengeyi kurabilmektedir.

Borealis de Aurora* ve Medusa'ya benzer bir yaklaşım önermiş ancak ek olarak aynı sunucu üzerinde yaşanacak yük çakışmalarının da önlenmesine yönelik iyileştirmelerde bulunmuştur [18]. Borealis, Aurora projesinin üzerine inşa edilen yeni bir dağıtık mimaride akan-veri işleme motorudur. Bu yapısıyla hem sensörler gibi küçük hem de sunucular gibi büyük ölçekteki veri işleme birimlerine yayılabilecek tek ve ortak bir altyapı kurmaktadır. Flux'da olduğu gibi, aşırı yük tespit edildiğinde, en çok yüklü olan makineden en az yüklü olan makineye veri aktarımı yapılmaktadır. Önerilen yöntem ile sadece makineler arasındaki yük dengelenmemekte, aynı zamanda her makinenin üzerinde oluşan yük değişimi de minimumda tutulabilmektedir. Yöntemle, yoğunluk oluşan sunucu üzerindeki verilerin diğer sunuculara aktarılmasıyla küresel ölçekte yükün dengelenmesi hedeflenmiştir. Bu da tek bir yönetim altında toplanan makinelerin birbirileriyle tam iletişim ve iş birliği içerisinde olmalarıyla başarılmıştır.

Gedik ise dağıtık mimarilerde durumlu akan-veriler üzerinde çalışacak yeni bir bölüntüleme yöntemi geliştirmiştir [19]. Bu yöntem ile işlenen verinin sıklıkları hesaplanarak makineler arası aktarım maliyeti ve sistemdeki yük dengesizliği kontrol altına alınmaya çalışılmaktadır. Yöntem, verinin çarpık olması durumunda dahi, işlem, iletişim ve bellek kullanımlarını dengeleyebilmekte ve veri aktarım maliyetini minimumda tutabilmektedir.

Benzeri şekilde **Balkesen** de gelen verinin sıklığını hesaplayarak yükü makineler arasında dengelemeye çalışmaktadır [20]. Bu kapsamda sadece geçmiş verilere bakmayarak, aynı zamanda gelecek verileri de tahmin etmeye çalışarak uyarlanabilen veri dağıtımı sunmaktadır. Bunu da sisteme giriş yapacak veriler hakkında tanımlayıcı bilgileri ve yapılacak sorgularla ilgili gereksinimleri alarak verilerin sisteme giriş yapacakları noktaları belirleyerek yapmaktadır. Yöntemle hedeflenen veri akışını ve makinelere atanmasını sistemin haricinden kontrol ederek, akan-veri işleme araçlarının daha dinamik ve performanslı çalışmalarını sağlamaktır.

Bir başka çalışmada ise **Fernandez**, durumlu işlemlerin yapıldığı sistemlerde, işlemlere ait durum bilgilerinin dışa çıkartılarak yönetilmesini önermektedir [21]. Bu yöntemde sistemin dinamik olarak yatay büyüyebilmesi ve aynı zamanda durumlu işlemlerde hata dayanıklılığının sağlanarak kurtarma senaryolarının uygulanabilmesi için tümleşik bir yaklaşım sergilenmektedir. Dışa çıkartılan durum bilgileri belirli aralıklarla kontrol noktası olarak kaydedilmektedir. Bu sayede, sistem bir tıkanıklık veya yük dengesizliği tespit ettiğinde ya da makinelerden birinin işlem yapamaz hale gelmesi durumunda bu kontrol noktalarını kullanarak kalan işleri diğer makinelere dağıtabilmekte ve hem sistemin yatay olarak genişleyebilmesini hem de hatalara karşı dayanıklı olabilmesini sağlayabilmektedir.

Yukarıda belirtilen çalışmalar ya merkezi bir sistem tarafından yönetilmesi gereken yapılar içermekte ya da makineler arası veri aktarımının yapılmasını gerekli kılmaktadır. Her ikisi de dağıtık mimarilerde gerçek zamanlı sonuç üretebilmemiz için yeterli olamamaktadır. Çünkü dağıtık mimarilerde, makinelerin sürekli birbirleriyle iletişim içinde olarak büyük hacimli verileri birbirlerine aktarmaları sistemin bütününün performanslı çalışmasını olumsuz etkilemektedir ve yüksek ağ trafiği yaratmaktadır.

Aynı zamanda, merkezi bir yönetimin olması da olası hata durumlarında bütün sistemin durmasına ve iş yapamaz hale gelmesine yol açmaktadır. Sistemin hem hatalara karşı dayanıklı hem de yüksek performanslı şekilde çalışmasını sağlayabilecek bir yapıda tasarlanması gerekmektedir. Bu nedenle merkezi bir yönetime ve makineler arası veri aktarımına ihtiyaç duymayan bir çözüm gereksinimi oluşmaktadır. Bu kapsamda, mevcut çalışma dahilinde en çok faydalanılan ve geliştirilmeye çalışılan yöntem ise PoTC (The Power of Two Choices) olmuştur [22]. **Azar** tarafından tanıtılan PoTC yöntemi, yük parçalarını makinelere atayarak dengelemeyi başaran basit ve etkili bir tekniktir ve en etkili, toplar-ve-sepetler (balls-and-bins) yaklaşımı ile açıklanabilmektedir. Daha önce de belirtildiği üzere, bu yaklaşımda, yapılacak işler top, işi yapacak makineler de sepetler olarak kurgulanmaktadır. Tek tercihin yapıldığı yaklaşımda, her top için rastgele bir sepet seçilmekte ve aynı top hep aynı sepete atılmaktadır. Öte yandan, iki tercihin yapıldığı yaklaşımda, her top için rastgele iki sepet seçilmekte ve toplar hep seçilen bu iki sepet arasından en az yoğun olana atılmaktadır. Bu yöntemle, gerçek zamanlı yük dengeleme konusunda diğer yöntemlere kıyasla oldukça başarılı sonuçlar alınabilmiştir.

Mitzenmacher de yük dengeleme algoritmaları ve PoTC yöntemi üzerine çalışmalarda bulunmuştur [23, 24, 25]. Bu çalışmalarında süpermarket modeli adını verdiği bir sistemi tanıtmıştır. Bu model, Azar'ın çalıştığı statik yük dengeleme yönteminin genişletilmesi olarak görülebilir. Model, sunucu sayısının sonsuzluğa ulaştığı sonsuz boyutlu bir sisteme karşılık gelen ideal bir sürecin tanımlanmasına odaklanmakta olup, iki hedef makine arasından seçim yapmaya olanak tanıyan basit bir dinamik yük dengeleme modelinde, rastgele düzgün bir şekilde dağılımın sistemin performansında üssel bir gelişim sergilediğini de göstermiştir.

Bu çalışma dahilinde ise, en çok, **Morales**'in The Power of Both Choices adlı çalışmasından faydalanılmış ve önerilen yöntem geliştirilmeye çalışılmıştır. Morales yaptığı çalışmalarda PoTC yaklaşımını temel alarak, dağıtık mimariye sahip ve akan-veri işleyen motorların üzerindeki yük dengeleme problemini incelemiştir [26]. Yöntemde, iki tercihin yapıldığı durumda kazancın, teorik olarak, tek tercihin yapıldığı duruma kıyasla üssel olduğu belirtilmektedir. Bununla beraber, ikiden fazla tercih

yapılması durumunda üssel bir kazanç sağlamayacağı belirtilmiştir. Bu nedenle yöntem dahilinde her veri için iki tercihin yapılmasıyla yetinilmiştir.

Morales'in bu çalışmasında, özellikle çarpık verilerin işlendiği durumlarda daha dengeli yük dağıtımı yapabilen PKG (Parçalı Anahtar Gruplama) yöntemi önerilmiştir. Bu yöntem ile üretkenlikte %60'a, gecikme süresinde ise %45'e varan iyileşmenin gözlemlendiği belirtilmiştir.

PKG yöntemi, iki temel teknik üzerine kurgulanmıştır: anahtar bölüntüleme ve yerel yük tahmini (local load estimation). Anahtar dağıtımı için PoTC (Power of Two Choices) yönteminden yararlanılmıştır. Bu yöntemde sistem, her bir anahtar için iki hedef makine belirlemekte ve mesajı aralarında en az yoğunluğa sahip olana yönlendirmektedir. Yalnız bunu yaparken, bütün kaynakların, seçilen hedef makine üzerinde el sıkışabilmesi adına, kaynakların hangi mesajı hangi makineye gönderdiklerinin bilgisini tutmaları gerekmektedir. Bu da sisteme ek bir konfigürasyon yükü getirmektedir. Öte yandan hedef makinelerin yük bilgilerini de bilebiliyor olmamız gerekmektedir. Geleneksel yaklaşımda bütün makinelere erişim yapabilen ve makinelerin yük durumlarını küresel ölçekte bilen sistemler mevcuttur. Ayrıca bu sistemlerde verilerin tek bir kaynaktan geldiği varsayılmaktadır. Ancak, dağıtık sistemlerde, birçok makineden oluşan bilgisayar kümelerinde çalışan ve birçok kaynaktan veri alan yapılarda, geleneksel yapıda olduğu gibi yük durumların bilinebilmesi çok daha zor olmaktadır. Genel ölçekte yük dağılımının bilinmesi zor olduğundan, bu makalede yerel yük tahmini tekniği kullanılmış ve pratikte oldukça tatmin edici sonuçlar alınmıştır. Bu teknikte, dağıtımı yapacak her birim, dağıtım yapacağı makinelerin yük bilgisini kendisi takip etmektedir. Bunu da makinelere yönlendirdiği mesaj sayısını tutarak yapmaktadır. Bu teknikle, genel ölçekte yük bilgilerine sahip olan geleneksel sistemlerle hemen hemen aynı sonuçların alındığı belirtilmiştir.

Çalışma kapsamında twitter, wikipedia gibi gerçek veri kümelerinin yanı sıra, sanal olarak ürettirilmiş yapay veri kümeleri de kullanılmış olup birçok İşçi Birim ve Kaynak Birim sayılarıyla deneyler yapılmıştır. Özellikle çarpık veri kümelerinde, KG yöntemine nazaran üretkenlik ve gecikme süreleri ele alındığında %45'e varan iyileşmeler tespit edilmiştir. Sonuç olarak, PKG yönteminin SG yönteminin kullanıldığı her yerde bellek

kullanımını azaltmak adına, KG yönteminin kullanıldığı yerlerde ise daha iyi yük dağıtımına sahip olabilmesi, üretkenlik ve verimliliğin arttırılması adına kullanılabileceği belirtilmiştir.

3.4. Hedef Makine Belirleme

Verilerin gönderilecekleri makinelerin belirlenmesi yük dağıtım algoritmalarının temel işlevini oluşturmaktadır. Temel olarak iki farklı yöntem mevcuttur: birincisi, verinin içeriğine bakılmaksızın karar verilen SG yöntemiyken, diğeri ise KG temelli olan ve tercihin verinin içeriğine göre yapıldığı yöntemdir.

SG yönteminde, Round-Robin yaklaşımı kullanılarak, gelen veriler içeriğine bakılmaksızın, uygun makinelere sırasıyla dağıtılmaktadır. Verinin içeriğinin önemli olduğu KG temelli yöntemlerde ise verinin içerisindeki anahtar değerin karma değeri hesaplanır ve sistemde mevcut makine sayısına göre mod işlemi hesaplanarak hedef makine belirlenir. Bu sayede, aynı anahtar değerine sahip bütün verilerin aynı makineye gönderilmesi garantilenmiş olur.

$$\boldsymbol{Tm = H1(data)\ \%\ Nm}$$

Tm: Hedef Makine

Nm: Tercih Edilebilecek Makine Sayısı

Özetle bütün yöntemlerin yapmaya çalıştığı, sistemdeki yükü makineler arasında eşit şekilde paylaşarak, çıktı üretkenliğini arttırmak, çıktı süresini düşürmek ve sisteme giriş yapan verinin en kısa sürede işlenerek sistemden çıkmasını sağlamaktır. Verinin sistemden anlamlı bir biçimde çıkabilmesi içinse, farklı makinelerde işlenen ama tek başlarına bir anlam ifade etmeyen durumlu verilerin, ilişkilendirilmesi ve bütünleştirilerek anlamlı tek bir çıktı üretmesi sağlanmalıdır. Burada ise dağıtım maliyeti faktörü, yani ilişkilendirme/bütünleştirme maliyeti devreye girmektedir. İlişkili veriler ne kadar çok farklı makinelere dağıtılırsa, dağıtım maliyeti değeri de o kadar yüksek olmaktadır. Çünkü anlamlı bir çıktı üretilebilmesi için bir yerine birçok makinenin çıktısı beklenmekte ve ancak toplanan bu verilerin tamamının üzerinde bir işlem gerçekleştirildikten sonra anlamlı bir çıktı üretilebilmektedir.

Örnek olarak, bin sayfalık bir kitabın içerisinde geçen kelimelerin sıklıklarının çıkartılmaya çalışıldığını düşünelim. Mevcut olan yüz adet makineye bu işi en kısa sürede yaptırmak ve sonuç üretmek esas gereksinimi oluşturmaktadır. Bu kapsamda yükün, yani kitabın içerisinde geçen kelimelerin, sistemdeki makinelere dağıtılması gerekmektedir. Kelimeler makinelere dağıtıldıktan sonra ise, her kelimenin ne kadar geçtiğini öğrenebilmek için, farklı makinelerdeki işlenen aynı kelimelerin birleştirilmesi gerekmektedir ki nihai sonuç oluşturulabilsin. Yani "bilgisayar" kelimesi 1. makinede 10 kez geçerken, 2. makinede 15 kez ve 3. makinede 5 kez geçiyorsa, toplam 30 kez geçiyor diyebilmek için bu 3 makinedeki sonuçların birleştirilmesi ve tek çıktı üretilmesi gerekmektedir. Aksi takdirde hiçbir makinedeki veriler tek başına yeterli ve anlamlı olamamaktadır. Bu noktada, SG yöntemini kullanılırsa, yük en iyi şekilde dağıtılmış olacaktır ancak; her makinede her kelime geçeceğinden, anlamlı çıktı üretebilmek için bütün makinelerin çıktılarını beklemek ve sonra birleştirmek gerekli olacaktır (yüksek Dağıtım Maliyeti, düşük Gecikme). KG yöntemi kullanılır ise, her kelime sadece bir makinede işleneceği için, herhangi bir birleştirme maliyeti olmadan (düşük Dağıtım Maliyeti) ve başka bir makinenin çıktısı beklenmeden, doğrudan anlamlı çıktı üretilebilecektir. Ancak bu senaryoda düşük Gecikme garanti edilememektedir. Çünkü KG yöntemi verinin içeriğine doğrudan bağlıdır. Yani daha sık geçen kelimeler belirli makinelerde yoğunluklar yaratacaktır. İstatistiksel olarak bir kitapta en çok "ve" kelimesinin geçtiğini göz önünde bulundurduğumuzda, KG yöntemi ile 100 makine arasında bir tanesinin diğerlerine kıyasla daha yoğun olması kaçınılmaz olacaktır.

SG yöntemi her veriyi her makineye göndererek yüksek dağıtım maliyetine yol açarken, KG yöntemi her bir veri için tek bir hedef makine belirleyerek çarpık veri kümesiyle işlem yaptığı durumlarda bazı makinelerin aşırı yüklenmesine sebebiyet vermekte ve üretkenliği azaltarak çıktı süresini uzatmaktadır. İlişkisel verilerin bir arada tutulması gerektiğinden SG yöntemi tercih edilememektedir. KG yöntemi de tek tercih yaptığı için veri içeriğine bağlı olarak öngörülebilir yük dağılımına ve gecikme süresine sahip olunmasını engellemektedir. Tercih sayısının 1 ile sınırlı kalmayarak, her veri için 2 adet makinenin tercih edilmesi, KG yönteminin aksine, yükün daha dengeli dağıtılabilmesini ve gecikme süresinin ise azalmasını sağlamaktadır [6]. Parçalı Anahtar Gruplama (PKG) adı verilen bu yöntem her bir veri için 2 tane makine tercihi

yapmakta ve yük dağılımını yerel olarak tutarak, ikisi arasında en az yoğunluğa sahip olanı tercih etmektedir. Bu sayede, üretkenlikte ve gecikme sürelerinde %45'e kadar iyileştirme sağlandığı belirtilmektedir.

PKG yönteminde, KG yönteminin aksine 2 adet karma fonksiyonu belirlenmekte ve aynı veri için 2 farklı değer üretilmesi sağlanmaktadır. Bu sayede her veri için 2 hedef makine belirlenmiş olmaktadır. Sonrasında bu iki makineden daha az yoğunluğa sahip olan hedef makine olarak belirlenmektedir. Yük bilgileri ise her makine üzerinde yerel olarak tutulmaktadır. Bu sayede ortak ve merkezi bir makineye ihtiyaç duymadan karar verebilmek için yeterli bilgiye sahip olunmaktadır.

$$\boldsymbol{M1 = H1(data)\ \%\ Nm}$$

$$\boldsymbol{M2 = H2(data)\ \%\ Nm}$$

$$\boldsymbol{Tm = min(L(M1), L(M2))}$$

L: Makinenin yükü

Dikkat edileceği üzere, bu yaklaşımda yükün her daim iki makineye dağıtılacağının garantisi verilememektedir. Çünkü karma fonksiyonlarının ürettiği değerler farklı olsa dahi, mod alındıktan sonra çıkacak dizin değerleri aynı olabilmektedir. Özellikle hedef makine sayısının az olduğu durumlarda, bu çakışmaların oluşma ihtimali daha da artmaktadır.

4. ÖNERİLEN YAKLAŞIM

Mevcuttaki AVİ yöntemleri birçok problemde başarılı sonuç üretebilmektedir. SG yöntemi ilişkisiz veri kümelerinde ve KG yöntemi homojen veri kümelerinde verimli olurken, PKG ise heterojen veri kümelerinde güzel sonuçlar üretebilmektedir. Ancak bütün bu yöntemler, verinin yüksek ölçüde çarpık geldiği durumlarda başarısız olmaktadır. Verilerin ilişkili olduğu durumlar için SG yöntemi değerlendirmeye alınmamaktadır. Öte yandan KG yöntemi, %80 çarpık gelen bir veri kümesinde 10 makineden sadece 1'ini %80 yoğunlukla çalıştırıp geri kalan %20 yükü 9 makineye dağıtacaktır. PKG yöntemi yükü biraz daha dengeli dağıtmakta, ancak o da 2 makineyi %40 yükle çalıştırırken yine geri kalan %20 yük 8 makineye dağıtacaktır. Bu da demek oluyor ki çarpık veri kümesi işlenirken %20 verim alınabilecek ve çalıştırdığımız 10 makinenin 8 tanesi âtıl durumda bekleyecektir. Makine sayımızı 100'e çıkardığımızı düşünürsek; KG yöntemi ile 1 makinede %80 yük oluşurken, %20'lik yük 99 makineye dağıtılacaktır. PKG yönteminde ise, yine 2 makineye %40 yük dağıtılırken %20'lik yük 98 makineye dağıtılacaktır. Bu durumda verim %2'ye düşmüş olacaktır. Anlaşılacağı üzere, bu yaklaşımlarda sistemde yoğunluk tespit edildiğinde yatay büyümenin bize hiçbir faydası olmayacağı gibi, aksine maddi zararı olacaktır. Bu nedenle, bu çalışma kapsamında çalıştırılacak makine sayısının sabit olduğu varsayılıp, mevcut makinelerle, yatay büyüme yapmadan, yükü en dengeli şekilde dağıtabilecek ve özellikle çarpık veri kümelerinde başarılı yük dağılımı yapabilecek bir yöntem geliştirilmeye çalışılmıştır.

PKG yönteminde, KG yöntemindeki tek tercih ikiye çıkartılmış ve gözle görülür performans artışı gözlemlenmiştir [26]. Ancak, çarpıklığın arttığı veri kümelerinde PKG yöntemi de etkili yük dağılımı yapamamıştır. Biz ise, bu sayının 2 ile kısıtlı kalmayarak, verinin içeriğine doğrudan bağlı olarak değişmesi gerektiğini öngördük. Yani her veri için sabit 2 tercih yapmak yerine, veri ne kadar yoğun ve çarpık ise o denli çok tercih yapılabileceğini ve böylelikle daha dengeli yük dağılımı yapabileceğini varsaydık. Yoğun olmayan veriler için ise, PKG yönteminin öngördüğü gibi 2 tercihi varsayılan

olarak belirledik. Bu yönteme ise **Dinamik Anahtar Gruplama (DKG – Dynamic Key Grouping)** adını verdik.

DKG yönteminde, tercih sayısı 2 ile kısıtlı kalmadığından ve veri içeriğine bağlı olarak artabiliyor olduğundan bahsetmiştik. Bu kararı verebilmek içinse, çarpık gelen verileri tespit etmemiz, makinelerin yoğunluklarını tahmin edebiliyor olmamız ve zaman içerisinde dinamik bir şekilde bir anahtar kelimenin gidebileceği makine sayısını arttırıp azaltabiliyor olmamız gerekmektedir. Daha önce de belirtildiği üzere, bu çalışma kapsamında kümedeki makine sayısının değişmediği varsayılıp, mevcut makineler üzerinde yük dağılımının daha iyi yapılabilmesi adına önermelerde bulunulmaktadır. Aşağıda, çalışma kapsamında önerilen DKG yönteminin detayları verilmektedir.

DKG yönteminde, her anahtar kelimenin dağıtılabileceği makine sayısı başlangıçta varsayılan olarak 2'dir. Bunu her anahtar kelime için iki makine belirlenmiş olarak düşünebiliriz. Aslında, her anahtar kelimenin dağıtılabileceği makineler listesi bulunmaktadır ve sistem gelen yükü bu makineler arasında dengeli olarak dağıtmaktadır. Çarpık veya yoğun gelen veriler için zaman içerisinde bu listeye yeni makineler eklenebilir. Dolayısıyla, sistemin, yükü dağıtabileceği makine sayısı artmış olur. Yük azaldığında ise, bu listeden, sondan başlamak kaydıyla eklenen makineler çıkarılacaktır. Bu sayede, sistem dinamik olarak hem büyüyüp hem de küçülebilme yeteneğine sahip olabilmektedir.

DKG yöntemi, tanımlı bileşenlerini kullanarak, yoğun gelen verilerin tespitini yapmakta ve uygun koşulların oluşması durumunda yükü daha fazla makineye dağıtabilmektedir. DKG yöntemi de KG ve PKG yöntemlerinde olduğu gibi makinelerin yük dağılımlarını yerel olarak takip etmektedir. Yani sistemin tamamına hâkim olarak gerçek yük durumlarını bilememektedir. Bunun yerine, dağıtım yapacak her birim, her makineye sayısal olarak ne kadar dağıtım yaptığını tutmakta ve doğal olarak kendi içindeki yük dağılımlarını bilebilmektedir. Yerel yük dağılımı olarak da bilinen bu yöntemde makinelerin yük dağılımları kesin doğrulukla bilinememesine karşın, oldukça yakın ve yeterli sonuçlar alınabildiği gözlemlenmiştir [26].

DKG yöntemi tüm bu yetenekleri gerçekleştirebilmek için birtakım bileşenlerden ve algoritmalardan oluşmaktadır.

4.1. Sistem Bileşenleri

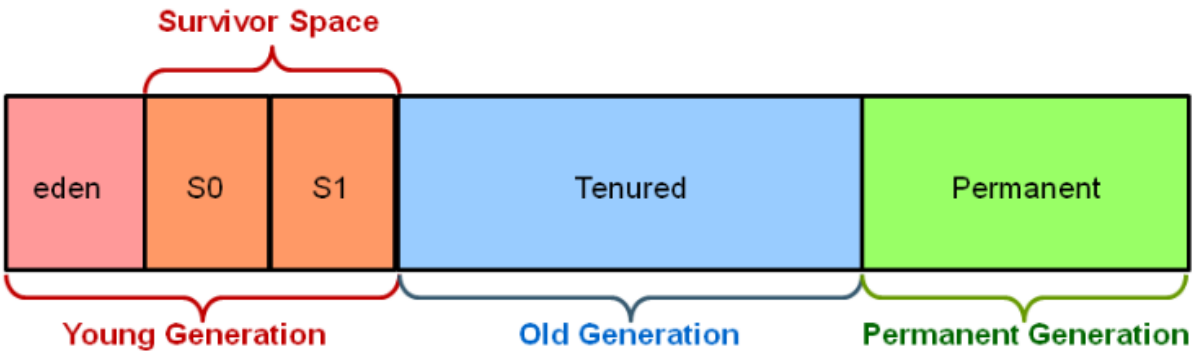
Bu kesimde, DKG yönteminin gerçekleştirimine ait bileşenler ve çalışma prensipleri anlatılacaktır.

4.1.1. Anahtar Birimi

Sistemde dolaşan ve işleme giren her anahtar kelime için bir Anahtar Birimi bileşeni oluşturulmaktadır. Bu bileşen, anahtar kelimeyi, sisteme en son ne zaman giriş yaptığını, toplam kaç tane geçtiğini ve bu anahtar kelime için en son ne zaman yatay büyüme kontrolü yapıldığını tutmaktadır. DKG yöntemi, bütün kararları ve uygulamaları Anahtar Birimi bileşenleri üzerinden yürütmektedir.

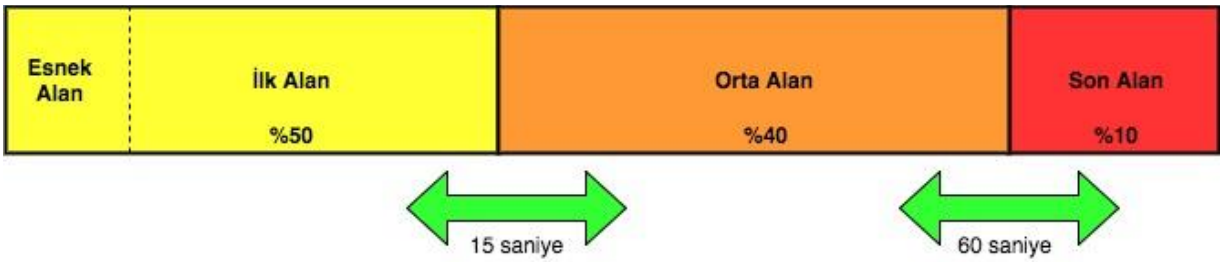
4.1.2. Anahtar Alanı

Anahtar Birimlerini yönetebilmek ve veri yoğunluklarını / çarpıklıklarını tespit edebilmek adına Anahtar Alanı adı verilen bir bileşen tasarlanmıştır. Bu bileşen tasarlanırken, Şekil.15'te gösterilen Java Sanal Makinesi (JVM) yığın modelinden esinlenilmiştir. JVM üzerinde oluşturulan nesneler, öncelikle Genç Alan (Young Generation) içerisindeki Cennet Alanında (Eden Space) yer alırlar. Cennet Alanı dolduğunda, Küçük Atık Toplama Döngüsü (Minor GC) çalıştırılır ve hayatta kalan, yani hala aktif olarak kullanılan nesneler, S0 ve S1 olarak ayrılan Yaşayan Alana (Survivor Space) aktarılırlar. Atık Toplama Döngüsü (Garbage Collecting), JVM tarafından kullanılmayan nesnelerin bellekte yer tutmaması için temizlenmesini ifade eder ve etkili bellek kullanımı için zorunludur. Her Atık Toplama Döngüsünde aktif olan nesneler S0 ve S1 Alanları arasında hareket ederek varlıklarını korurken, kullanılmayanlar ise JVM tarafından bellekten temizlenirler. Birçok GC döngüsünden sonra halen aktif olan nesneler ise bir sonraki evre olan Olgunlaşmış Alana (Old Generation) aktarılırlar. Bu alanda uzun ömürlü olan nesneler tutulmaktadır. Bu sayede çok sık kullanılan ve sürekli aktif olan nesnelerin tekrar yaratılma maliyetinin önüne geçilmiş olur. Olgunlaşmış Alan içerisinde yaşayan nesneler ise Büyük Atık Toplama Döngüsü (Major GC) denilen bir döngüde düzenli olarak taranır ve kullanılmayanları bellekten temizlenir. Kalıcı Alan (Permanent Generation) ise, üstveri olarak adlandırılan, uygulama içerisinde bulunan sınıf ve yordamlara ait bilgilerin tutulduğu bölgedir. Bu bölge herhangi bir GC döngüsüne tabi tutulmaz ve buradaki bilgiler uygulama boyunca aktiftir.



Şekil 15. JVM Yığın Modeli [67]

DKG Anahtar Alan modeli Şekil 16'da da görüleceği üzere, JVM Yığın Modeline benzer şekilde tasarlanmıştır. Anahtar Alanı, İlk Alan, Orta Alan ve Son Alan olmak üzere üç kısımdan oluşmaktadır. Bu kısımlarda Anahtar Birimi bileşenleri tutulmaktadır. Bu kısımların boyutları ise dinamik olarak belirlenmektedir. Yoğun olarak beklenen farklı anahtar kelime sayısı yaklaşık olarak tahmin edilip sisteme girilmektedir. Bu sayının %10'u Son Alan, %40'ı ise Orta Alan boyutu olarak belirlenmektedir. Geri kalan %50'lik kısım ise İlk Alan olarak belirlenmektedir. Bu çalışma kapsamında bu sayıyı 100 olarak belirledik ve Son Alan 10, Orta Alan 40 ve İlk Alan 50 Anahtar Birimi tutacak şekilde ayarlanmış oldu.



Şekil 16. DKG Anahtar Alan Yönetimi

Son Alan ve Orta Alanın fiziksel sınırları bulunurken, İlk Alanın fiziksel sınırı bulunmamaktadır. Yani Son Alan 11. Anahtar Birimini tutamayacağı gibi, Orta Alan da de 41. Anahtar Birimini tutamayacaktır; ancak İlk Alan istediği kadar Anahtar Birimini

tutabilecektir. Bu sayede, sisteme anlık olarak giriş yapan birçok verinin, bir üst kısma geçebilmesi için yeterli imkân ve zaman verilmiş olacaktır.

Sisteme giriş yapan her anahtar kelime için önce Son Alan, sonra Orta Alan ve en son İlk Alan kontrol edilir. Eğer ilgili kısımlarda bulunuyorsa, Anahtar Birim bileşeni getirilir ve üzerindeki en son görünme zamanı güncellenerek toplam kaç kere geldiğini gösteren sayaç arttırılır. Eğer bu kısımların hiçbirinde bulunamazsa yeni bir Anahtar Birimi bileşeni oluşturularak İlk Alana eklenir.

4.1.3. Anahtar Alan Yöneticisi

Anahtar Alan Yöneticisi ise, ayrı bir iş parçasında çalışarak, Anahtar Birimlerinin Anahtar Alanları arasındaki geçişini ve Atık Toplama Döngüsünü yönetmektedir. 15 saniyelik döngülerle Anahtar Alanını kontrol etmekte ve İlk Alandan Orta Alana yükselmesi gereken Anahtar Birimlerini tespit ederek gerekli geçişi gerçekleştirmektedir. Benzer şekilde, 60 saniyelik döngülerle ise Orta Alan ile Son Alan arasındaki Anahtar Birimi geçişi sağlanmaktadır. Anahtar Birimi geçişleri İlk Alandan Orta Alana ve Orta Alandan Son Alana doğru olabilmektedir. Geçiş yönünü betimlemek için Kaynak ve Hedef Alan kullanılmaktadır. İki Alan arasındaki Anahtar Birimi geçişi ise şöyle gerçekleşmektedir:

1. Sisteme sürekli veri girişi olduğundan bir Alan içerisindeki Anahtar Birimlerinin sayaçları sürekli artar ve yoğunluk sıralaması bozulur, bu nedenle öncelikle aralarında geçiş yapılacak Alanlar kendi içlerinde sayaçlarına bakılarak büyükten küçüğe sıralanır. Bu sıralama sonunda Kaynak Alan ve Hedef Alan kendi içerisinde yoğunluklarına göre sıralanmış olur.
2. Kaynak Alanın başındaki Anahtar Birimi ile Hedef Alanın en sonundaki Anahtar Biriminin sayacı karşılaştırılır. Eğer Kaynak Alandaki Anahtar Biriminin sayacı daha büyükse, bu Anahtar Birimi ile Hedef Alandaki Anahtar Birimi yer değiştirir.
3. Aynı işlem Hedef Alanın bütün Anahtar Birimlerini gezinceye kadar devam eder.
4. İşlem tamamlandığında, daha sık gelen Anahtarlar Hedef Alanda yer almış olur.

Atık Toplama Döngüsü ise İlk Alan ile Orta Alan arasındaki Anahtar Birimi transferinden sonra gerçekleşir. Daha yoğun olan Anahtar Birimleri Hedef Alana aktarıldıktan sonra,

İlk Alanın boyu başlangıçta belirlenmiş olan varsayılan değerine küçültülür. Yani, İlk Alanda 150 Anahtar Birimi yer alıyorsa, son 100 Anahtar Birimi silinerek ilk 50 Anahtar Birimi saklanır. Bu sayede belleğin sürekli büyümesinin önüne geçildiği gibi, sürekli olarak gelmeyen ve sistemde yoğunluk oluşturmayan Anahtarların elenmesi sağlanır.

İlk, Orta ve Son Alanlar arası geçişler yukarıda özetlendiği gibi, aynı zamanda aşağıda verilen Algoritmalar 1-3 ile de tanımlanabilir.

Algoritma 1, Anahtar Birimlerinin İlk Alandan Orta Alana geçişini vermektedir.

Algoritma 1: ortaAlanaGecis

Sonuç: Anahtar Birimlerinin İlk Alandan Orta Alana Geçişini Sağlar
Girdi: İlk Alan (İA), Orta Alan (OA)
Çıktı: -
İA içerisindeki Anahtar Birimlerini sırala
OA içerisindeki Anahtar Birimlerini sırala
İA'nın sonundaki Birimleri bellekten at (> 50)
digerAlanaGecis(İA, OA) yöntemini **çağır**
15 saniye bekle
ortaAlanaGecis(İA, OA) yöntemini **çağır**

Algoritma 2, Anahtar Birimlerinin Orta Alandan Son Alana geçişini vermektedir.

Algoritma 2: sonAlanaGecis

Sonuç: Anahtar Birimlerinin Orta Alandan Son Alana Geçişini Sağlar
Girdi: Orta Alan (OA), Son Alan (SA)
Çıktı: -
OA içerisindeki Anahtar Birimlerini sırala
SA içerisindeki Anahtar Birimlerini sırala
digerAlanaGecis(OA, SA) yöntemini **çağır**
60 saniye bekle
sonAlanaGecis(OA, SA) yöntemini **çağır**

Algoritma 3 ise, Anahtar Birimlerinin iki Alan arasındaki geçişini vermektedir.

Algoritma 3: digerAlanaGecis

Sonuç: Anahtar Birimlerinin Bir Alandan Diğer Alana Geçişini Sağlar
Girdi: Kaynak Alan (KA), Hedef Alan (HA)
Çıktı: -
Her bir HA içerisindeki Anahtar Birim **için;**
  Fi ← KA'daki ilk Anahtar Birim
  Fio ← Fi'deki Anahtarın toplam geçiş sayısı
  Li ← HA'daki son Anahtar Birim
  Lio ← Li'deki Anahtarın toplam geçiş sayısı
  Eğer Lio >= Fio **ise;**
    Döngüyü kır (geçiş tamamlanmış demektir)
  Aksi taktirde;
    Fi'yi KA'dan çıkart ve HA'nın başına ekle
    Li'yi HA'dan çıkart ve KA'nın sonuna ekle
  Bitiş
Bitiş

4.2. Çarpık Verilerin Tespiti

Anahtar Alanı yapısında Son Alan içerisinde yer alan Anahtarlar, DKG yönteminde çarpık veri olarak değerlendirilmektedir. Yani bir Anahtar çok sık geliyorsa, zaman içerisinde Son Alanda yer alacak ve uygulama da çarpık gelen bu Anahtarları ek makinelere dağıtabilecektir. Diğer bir deyişle, bir Anahtarın fazladan bir makineye gönderilebilmesi, bu Anahtarın içinde bulunduğu Anahtar Biriminin Son Alanda yer almasıyla mümkün olabilmektedir.

4.3. Eşik Değerlerinin Belirlenmesi

Bir Anahtarın Son Alanda olması, ek makinelere dağıtılabilmesi için tek başına yeterli değildir. Bunun yanı sıra, hem sistemin bir süre çalışıyor olması gerekmektedir (cold-start) hem de oluşan yoğunluğun belirli bir değere ulaşması gerekmektedir. Bu nedenle

eşik değerlerin belirlenmesi ihtiyacı doğmuştur. Yani, Son Alana dahil olmuş bir Anahtarın dağıtılabileceği makinelerin yoğunluğu belirlenen eşik değerini aşarsa, yeni bir makineye dağıtım yapılabilecektir.

Eşik değerinin belirlenebilmesi için öncelikle makineler için ideal yük değerini belirlememiz gerekmektedir. İdeal yükü, toplam yükün bütün makinelere eşit olarak dağıtılması olarak tanımlayabiliriz. Yük dağıtımını yüzde üzerinden karşılaştıracağımız için toplam yüke 100 dersek ideal yük bağıntısını şu şekilde verebiliriz:

$$\boldsymbol{Li = \frac{100}{m}}$$

Makinelerde beklediğimiz ideal yükü hesapladıktan sonra, sezgisel olarak belirlediğimiz eşik değerine ait bağıntıyı ise şu şekilde verebiliriz:

$$\boldsymbol{Lt = Li + \sqrt{Li}}$$

Bu sezgisel yaklaşımla, hem yükün yeni makinelere dağıtımını ideal yük üzerinden hesapladığımızdan toplam makine sayısına bağlamış oluyoruz, hem de oluşabilecek hafif yük artışlarında hemen yeni makine açılmasını engellemiş oluyoruz. Ayrıca, matematiksel olarak da bir anahtar kelime için seçilebilecek makine sayısını kısıtlamış oluyoruz. Bu sayede, uzun vadede daha kararlı ve daha performanslı bir sistem elde etmiş oluyoruz.

Çizelge 1'de, sistemde bulunan makine sayısına karşılık hesaplanan İdeal Yük ve Eşik Değeri verileri ile bir verinin en fazla kaç makineye dağıtılabileceği bilgisine de Maksimum Makine Sayısı altında yer verilmektedir. Örnek olarak, sistemimizde 10 makine olduğunu düşünürsek, makine başına ideal yükümüz %10 olmaktadır. Yeni bir makine seçebilmek için gerekli olan eşik yük miktarı ise %13.16 olarak hesaplanmıştır. Yani seçilebilecek makineler %13.16'dan daha fazla yüke sahipse, yük yeni bir makineye dağıtılabilmeye aday olmaktadır. Bununla birlikte, yük ne kadar fazla olursa olsun, en fazla 8 makineye kadar dağıtım yapılabilecektir. Yeni makineye dağıtım algoritmaları bir sonraki kesimde detaylı olarak aktarılacaktır.

Çizelge 1. Makine sayısına oranla yük eşik değerleri

Makine Sayısı	İdeal Yük (%)	Yük Eşik Değeri (%)	Makine Sayısı Limiti
5	20.00	24.47	5
10	10.00	13.16	8
20	5.00	7.24	14
50	2.00	3.41	30
100	1.00	2.00	51

4.4. Hedef Makinenin Belirlenmesi

Gelen her veri için anahtar kelime alınır ve bu kelimenin karma (hash) değeri hesaplanır. Karma değeri veriye özgü olan biricik bir tam sayıdır. Daha sonra bu değer aşağıdaki formül ile düzgelenerek hedef makinemizi belirler.

$$\boldsymbol{dizin} = \boldsymbol{karma}(\boldsymbol{anahtar}) \,\%\, \boldsymbol{N_m}$$

Varsayılan olarak her verinin dağıtılabileceği 2 makine olduğundan bu makineler n ve n+1 dizin değerlerine sahip olan makinelerdir. Dizin değerinin değiştiği her durumda düzgeleme işlemi tekrar uygulanır ki mevcut makinelerin dışına çıkılmasın (Ör. n+1 % N_m).

Verinin dağıtılabileceği makineler yerine, kaç tane makineye dağıtılabileceği bilgisi tutulur. Varsayılan olarak 2 olan bu değer, yoğunluğun olması durumunda sistemin genişlemesine ve sayının artmasıyla sonuçlanabilir. Her Anahtar için sistemde karşılık gelen bir sayı tutulmaktadır. Örneğin K1=4, K2=3, K3=6 gibi verinin dağıtılabileceği makine sayısı tutulmaktadır. Varsayılan 2 olduğundan, her değer için 2 sayısı tutulmaz, bunun yerine eğer sistemde anahtar kelimenin karşılık değeri yoksa 2 olarak alınır.

PKG yönteminde verinin dağıtılabileceği makineler iki farklı karma fonksiyonu kullanılarak sağlanmaktadır. Anahtar kelime için iki farklı karma fonksiyonunun sonucunda iki farklı dizin değeri oluşturulmaktadır. Bu iki dizin değerine karşılık gelen makinelerdeki yoğunluklar kıyaslanıp, az yoğun olan makine hedef olarak seçilmektedir. Burada önemli olan karma fonksiyonlarının nasıl tanımlandığıdır. Karma fonksiyonları verinin içeriğine sıkı sıkıya bağlı olduklarından, bir veri için iki farklı karma fonksiyonu aynı sonucu üretebilir. Bu durumda ise karma fonksiyonunun hiçbir yardımı

olmadığı gibi, yük dengeli dağıtılamayacağı için sistemde performans kayıpları oluşabilir. PKG yönteminde yükün bahsedildiği gibi en az 2 makineye dağıtılacağının garantisi verilememektedir, çünkü bu tamamen verinin içeriğine, yani belirlenen anahtar kelimeye ve sistemdeki toplam makine sayısına bağlıdır.

DKG yönteminde ise, karma fonksiyonu sadece ilk makine dizininin belirlenmesi için kullanılmaktadır. İkinci bir karma fonksiyonu yoktur. İlk dizin belirlendikten sonra gerekli olan diğer makinelerin dizin değerleri, bulunan ilk dizin değerini 1 arttırarak belirlenmektedir. Bu sayede yükün her zaman en az 2 makine arasında dağıtılacağı garanti edilmiş olur. Örneklemek gerekirse, 10 makineden oluşan bir sistemde K1 verisi için dizin 4 olduysa, başlangıçta 4 ve 5 numaralı makineler kullanılabilecektir. Daha sonra bu verinin yoğunluğu artarsa, sırasıyla 6, 7 ve 8 numaralı makineler de devreye alınabilecektir.

Hedef makinenin belirlenmesi aşağıda verilen algoritmada özetlenmektedir:

Algoritma 4: hedefMakineSec

Sonuç: Anahtarın dağıtılacağı hedef makine seçilir
Girdi: Anahtar, Son Alan (SA), Yatay büyüme eşik değeri (E)
Çıktı: hedef makinenin dizini
AK ← Anahtarın karması
HMD ← Hedef makine dizini düzgele(AK) yöntemi çağrılarak hesaplanır
Nm ← Anahtarın dağıtılabileceği makine sayısı (ön tanımlı olarak 2 seçilir)
LLmin ← Dağıtılabilecek makineler arasındaki en az yük miktarı
CM ← Şu ana kadar tespit edilen en iyi hedef makine
NLL ← Eşik değerinden düşük olan makine sayısı
enUygunMakineyiBul() yöntemini **çağır**
yatayBuyumeGerekliMi() yöntemini **çağır**
yatayKuculmeGerekliMi() yöntemini **çağır**
Eğer yatay büyüme gerekli **ise;**
    CMnew ← düzgele(HMD + Nm) yöntemini **çağır**
    LLnew ← CMnew makinesinin yerel yük bilgisi alınır

Eğer LLnew < LLmin **ise;**

CM ← CMnew

Nm ← Nm + 1

Bitiş

Yok Eğer yatay küçülme gerekli **ise;**

Nm ← Nm **–** 1

hedefMakineSec(Anahtar) yöntemini **çağır**

Bitiş

Mevcut CM değerini **döndür**

Yordam düzgele(dizin)

dizin % ToplamMakineSayısı sonucunu **döndür**

Bitiş

Yordam enUygunMakineyiBul()

Her bir hedef makine **için;**

cl ← makinenin yerel yük bilgisi

Eğer cl < T **ise**;

NLL ← NLL + 1

Bitiş

Eğer cl < LLmin **ise;**

LLmin ← cl

CM ← makine dizini

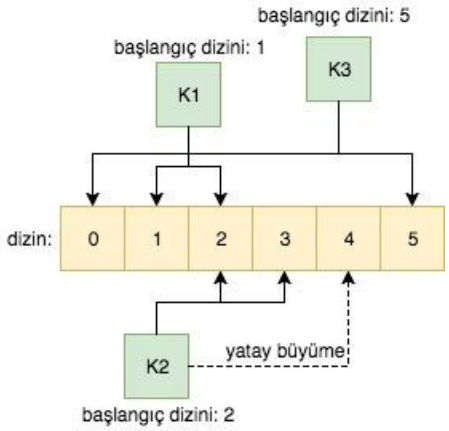
Bitiş

Bitiş

Bitiş

4.5. Yatay Büyüme

DKG yönteminde, sisteme yeni bir veri giriş yaptığında, verinin dağıtılabileceği makinelerin listesi hesaplanır. Sonra bu makinelerin yük durumları kontrol edilerek en az yüke sahip olan makine seçilir.



Şekil 17. Yatay büyümenin görselleştirilmesi

En az yüke sahip olan makine elde edildikten sonra, yeni makineye dağıtım yapılıp yapılmayacağı kararı verilirken şu kurallar işletilir:

1. Sistem yeni çalıştırıldıysa (cold-start) sistemin ısınması için belirlenen bir sürenin (15 saniye) geçmesi beklenmelidir. Bu süre zarfında sisteme çok fazla veri gelmediğinden sağlıklı karar verebilmek çok mümkün olamamaktadır. Bu nedenle sistem ısınmadan yeni makinelere dağıtım yapılamaz.
2. Yeni makine tercihinin yapılabilmesi için, mevcut makineler arasında tespit edilmiş olan en az yükün, sistemdeki makine sayılarına bağlı olarak Kesim 4.3'de belirlenmiş olan eşik değerlerinin üzerinde olması gerekmektedir. Aksi takdirde yükün yeterince çok olmadığı anlaşılmaktadır ve mevcut makinelerin bu yükü kaldırabilecek kapasitede oldukları düşünülmektedir.
3. Yeni bir makinenin seçilebilmesi için 2. maddede belirtilen yük eşik değerlerinin de aşılması tek başına yeterli değildir. Yükün dağıtımı için yeni makine seçilecekse bunun için gelen verinin Son Alanda yer alması beklenmektedir. Çünkü DKG yöntemi çarpık ve yoğun gelen verilere odaklandığından, sadece yoğun şekilde gelerek Son Alana geçmiş olan veriler için yatay büyüme

sağlanmaktadır. Bu sayede, çok sık gelmeyen veriler için gereksiz büyümenin önüne geçilmektedir.

4. 2. ve 3. maddelerde belirtilen kuralları sağlayarak yeni makine açılmasına karar verildiği durumda, gelen veri için kullanılabilecek yeni bir makine aday olacaktır. Kullanılabilecek olan yeni makinenin yük bilgisi de alınarak mevcutta hesaplanmış olan en az yüke sahip makinenin yükü ile kıyaslanmaktadır. Eğer yeni seçilen makinenin yükü daha az ise, sisteme bu veri için bir makine daha eklenecek ve gelen veri için kullanılabilecek makine sayısı 1 arttırılacaktır. Aksi takdirde yeni makine açmanın sisteme bir faydası olmayacaktır. Bu durumda yeni makine kullanılmayacaktır ve sistemde bir değişiklik yaratmayacaktır.

Yukardaki maddelerde belirtilen kurallar çerçevesinde yeni bir makinenin seçilemediği durumda, mevcutta seçilmiş olan en az yüke sahip olan makine kullanılacak ve veri bu makineye gönderilecektir.

Yukarıda verilmiş olan akış aşağıdaki yönergelerde özetlenmektedir:

Algoritma 5: yatayBüyümeGerekliMi

Sonuç: Yatay büyümenin gerekliliğini kontrol eder
Girdi: Anahtar, En Az Yük (Y), Eşik Değeri (E), Son Alan (SA), Çalışma Zamanı (ÇZ)
ÇZ ← Sistemin başlangıçtan itibaren ki çalışma zamanı
Çıktı: Sonucun mantıksal değeri
Eğer ÇZ <= 15 **ise**;
  yatay büyüme gerekli değildir sonucunu **döndür**
Yok Eğer Y < E **ise;**
  yatay büyüme gerekli değildir sonucunu **döndür**
Yok Eğer Anahtar SA içerisinde değil **ise;**
  yatay büyüme gerekli değildir sonucunu **döndür**
Aksi taktirde;
  yatay büyüme gereklidir sonucunu **döndür**
Bitiş

4.6. Yatay Küçülme

DKG yönteminin dinamik olabilmesi ve yükün azaldığı durumda fazladan aldığı makineleri sisteme geri verebilmesi için sistemin yatay küçülmeyi destekliyor olması gerekmektedir. Bu sayede Standart Sapma ve Dağıtım Maliyeti değerlerinde iyileşme gözlemleyebilir ve sistemin performansını arttırabiliriz.

Yatay büyümenin aksine yatay küçülme daha kolay belirlenebilmektedir. Gelen veri için ideal makine seçilirken, dağıtılabilecek mevcut makinelerin hepsi gezilerek yük durumları kontrol edilmektedir. Bu esnada, makinelerin yükleri, kesim 4.3'de belirlenen eşik değeri ile kıyaslanır ve eşik değerinin altında kalan toplam makine sayısı hesaplanır. Eğer verinin dağıtılabileceği makine sayısı 2'den fazla ise (en az 2 makinenin kullanılacağı garanti ediliyor) ve eşik değerinin altında kalan makine sayısı da 2'ye eşit veya daha fazla ise sistemin yatay küçülmesine izin verilecektir. Yatay küçülme kararı alındıktan sonra ise, verinin dağıtılabileceği makine sayısı 1 azaltılmakta ve hedef makine belirleme algoritması tekrar çalıştırılmaktadır. Bu sayede eğer sistemde birden fazla yatay küçülme yapılması gerekiyorsa bu hemen yapılabilecek ve sistemin performansı daha çabuk artış gösterecektir.

Yukarıda bahsedilen akış aşağıdaki yönergelerde özetlenmektedir:

Algoritma 6: yatayKüçülmeGerekliMi

Sonuç: Yatay küçülmenin gerekliliğini kontrol eder
Girdi: WC, NLL
WC ← Anahtarın dağıtılabileceği makine sayısı
NLL ← Eşik değerinden az yüke sahip dağıtılabilecek makine sayısı
Çıktı: Sonucun mantıksal değeri
Eğer WC > 2 ve NLL >= 2 **ise;**
yatay küçülme gereklidir sonucunu **döndür**
Aksi taktirde;
yatay küçülme gerekli değildir sonucunu **döndür**
Bitiş

4.7. Dağıtımın Gözlemlenmesi

DKG yönteminin ve diğer yöntemleri karşılaştırabilmemiz için yük dağılımlarını ve bu dağılım sonucunda meydana gelen gecikme süresi, üretkenlik, dağıtım maliyeti ve dağıtımın standart sapmasını gözlemleyebiliyor olmamız gerekmektedir. Bu kapsamda Gözlemci adı verilen birim, İşçi Birimleri dinlemekte ve bütün İşçi Birimlere gelen verileri ve bu verilerin makinelere nasıl dağıtıldığını takip etmektedir. Gözlemci Birimi ilingeye tamamen dışarıdan dahil olmakta ve istenildiği zaman kapatılabilmektedir. Bu birim, gerçek sistemlerde kullanılmayacak olup önerilen yöntemin ve diğer yöntemlerin karşılaştırılabilmesi için kullanılmıştır.

Gözlemci Biriminin topladığı verilerden karşılaştırma yapabileceğimiz 5 farklı alan ölçülmüş ve hesaplanmıştır. Bu alanlar ile ölçümlerine ve hesaplanmalarına ilişkin detaylar şu şekildedir:

- **Toplam Kayıt Sayısı:** Uygulamanın işlediği veriler içerisindeki toplam anahtar sayısını belirtir. Bu bilgi doğrudan Gözlemci Biriminin topladığı veriler üzerinden ölçülmektedir.
- **Gecikme (milisaniye):** Uygulamanın bütün verileri işlemesi için gerekli olan süreyi ifade eder. Uygulamanın verileri işledikten sonraki ilk çıktıyı üretmesi ile son çıktıyı üretmesi arasındaki süre ölçülerek hesaplanır.
- **Üretkenlik (kayıt/saniye):** Uygulamanın birim zamandaki çıktı miktarını yani üretkenliğini ifade etmektedir. İşlenen veri miktarının, geçen süreye bölümü ile hesaplanır. Geçen süre milisaniye cinsinden ölçüldüğünden öncelikle 1000 ile bölünerek saniyeye çevrilir, sonrasında saniyede üretilen iş miktarı aşağıdaki formül ile hesaplanabilir.

$$\mathbf{Th} = \mathbf{Toplam\ Kayıt\ Sayısı} / (\mathbf{Gecikme} / \mathbf{1000})$$

- **Standart Sapma:** Bir makine üzerindeki yükün, bütün makineler üzerindeki ortalama yüke bağlı yayılımını ifade eder. Diğer bir ifadeyle, yükün makineler arasında ne kadar dengeli dağıtıldığını gösterir. Standart sapmanın düşük olması yükün daha dengeli dağıtıldığı anlamına gelmektedir. Standart sapmanın hesaplanabilmesi için öncelikle her makine üzerindeki yükün aritmetik

ortalaması bulunur. Daha sonra her bir yük ile aritmetik ortalama arasındaki fark bulunur. Bulunan farkların her birinin karesi alınır ve elde edilen sayılar toplanır. Son olarak elde edilen sayı toplam makine sayısına bölünür ve karekökü alınır. Yapılan bağıntı aşağıdaki verilmiştir.

$$L_{avg} = \frac{\sum_{i=0}^{N_m} L_i}{N_m}$$

$$L_{SD} = \frac{\sqrt{\sum_{i=0}^{N_m} (L_i - L_{avg})^2}}{N_m}$$

- **Dağıtım Maliyeti:** Bir verinin kaç farklı makineye dağıtılmış olduğunu göstermektedir. Bir veri, ne kadar çok makineye dağıtılırsa, nihai sonucun çıkması için o kadar fazla makineden sonuç beklenecek ve o kadar fazla bütünleştirme işlemi yapılacaktır. Bu nedenle bu değerin yüksek olması sistemin verimsiz çalışması anlamına gelmektedir. Dağıtım Maliyeti, makinelerdeki toplam anahtar kelime sayısının, toplamdaki farklı anahtar kelime sayısına bölümü ile hesaplanmaktadır. Dağıtım Maliyeti en düşük 1 değerini alabilirken, alabileceği en yüksek değer ise sistemdeki İşçi Birim sayısı kadardır.

$$L_{DC} = \frac{\sum_{i=0}^{Nm} DK_i}{N_m}$$

SG, yükü en dengeli dağıtan yöntem olduğundan en düşük Standart Sapma değerine sahip olurken, her veriyi her makineye dağıttığından en yüksek Dağıtım Maliyeti değerine sahip olmaktadır. Öte yandan, KG ise, yükü verinin içeriğine bakarak dağıttığından en düşük Dağıtım Maliyeti değerine sahip olurken, verinin dağılımına göre yüksek Standart Sapma değerlerine sahip olmaktadır. PKG, KG yöntemine kıyasla daha iyi dağıtım yaptığından daha yüksek Dağıtım Maliyeti ve daha düşük Standart Sapma değerine sahip olmaktadır. PKG yöntemine alternatif olarak önerilen DKG ise, verinin içeriğine göre daha yüksek ve daha düşük değerler gösterebilmektedir. Yöntemlerin veri türlerine ve içeriklerine göre sahip oldukları

Standart Sapma ve Dağıtım Maliyeti değerleri Kesim 6'daki deney çıktılarında detaylıca incelenebilir.

5. ORTAM BİLEŞENLERİ

Bu kesimde yöntemlerin çalıştırabilmesi için gerekli olan ortam bileşenleri ve nasıl kurulacakları açıklanacaktır. Deney kapsamında Centos işletim sistemi üzerinde, Confluent, InfluxDB, Grafana bileşenleri kullanılmıştır.

5.1. Centos

Deney 64-bit Centos 6 [68] işletim sistemi üzerinde gerçekleştirilmiştir. İşletim sisteminin bir kopyası indirilerek CD veya FlashDisk'e kaydedilmeli ve sonrasında deneyin yapılacağı bilgisayara kurulmalıdır. Kurulum esnasında deneyin akışını etkileyecek özel bir yapılandırma yapılmamıştır. İşletim sistemi temel gereksinimleri karşılayacak şekilde istenildiği gibi kurulabilir. İşletim sistemi kurulduktan sonra aşağıdaki adımlar çalıştırılmalıdır:

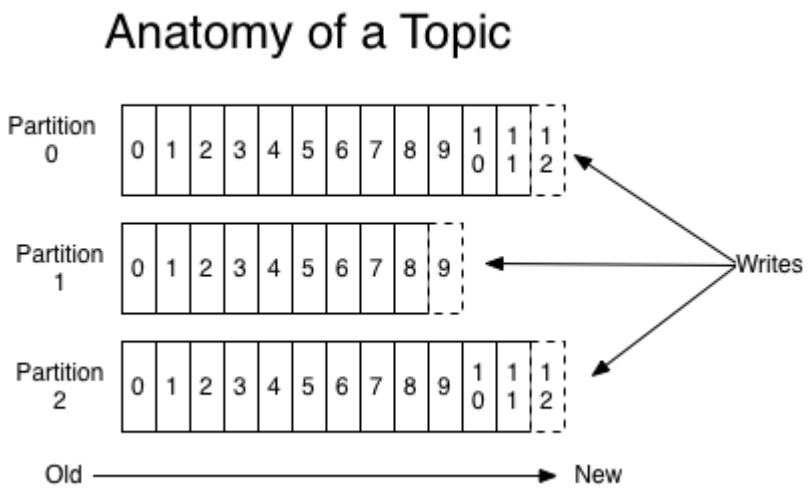
```
# işletim sistemi güncelleme
yum check-update && update -y

# java yükleme
wget --no-check-certificate --no-cookies --header "Cookie:
oraclelicense=accept-securebackup-cookie" http://download.oracle.com/otn-
pub/java/jdk/8u112-b15/jdk-8u112-linux-x64.rpm
yum -y localinstall jdk-8u112-linux-x64.rpm

yum install -y vim git
```

5.2. Confluent / Apache Kafka

Confluent [69], Apache Kafka üzerine inşa edilmiş, gerçek zamanlı veri işleme yapabilen açık kaynak kodlu bir platformdur. İçerisinde Zookeeper ve Kafka Connector gibi kullanmamız gereken bütün programları bir arada bize sunar.

Kafka mimarisi Publish-Subscriber modeline dayanır. Mesajlar konu adı verilen başlıklar altında kuyruk yapısında toplanır. Kafka, konuları diske yazarak kaybolmamalarını garanti eder. Her konu birçok bölüntüden oluşabilir. Bölüntü

sayısının artması aynı anda yapılabilecek okuma ve yazma işlemlerinin artmasını sağlayarak performansı artırabilir. Kafka, yapısal olarak dağıtık mimaride çalışabilmektedir. Bölüntüler Kafka kümesindeki makinelere dağıtılırlar ve *replication factor* ile doğru orantılı olarak tekrarlı saklanırlar. Bu sayede olası bir makine kaybına karşı dayanıklılığa sahip olurlar.



Şekil 18. Kafka mimarisi [8]

Confluent, resmî sitesinden indirilmeli ve ilgili dizin altına açılmalıdır. Kafka'ya ait sunucu ayarları yapıldıktan sonra, aşağıdaki adımlara izlenerek Confluent çalışmaya hazır olacaktır.

```
# install & configure
wget http://packages.confluent.io/archive/3.2/confluent-3.2.0-2.11.tar.gz
tar -zxf confluent-3.2.0-2.11.tar.gz
vim $CONFLUENT_HOME/etc/kafka/server.properties
* advertised.host.name=192.168.1.39
* zookeeper.connect=192.168.1.39:2181


# environment
export CONFLUENT_HOME=/home/programs/confluent-3.2.0
export KAFKA_HOME=$CONFLUENT_HOME
export KAFKA_PROPS=$KAFKA_HOME/etc/kafka
export KAFKA_HEAP_OPTS="-Xms1g -Xmx8g"
alias klist="$KAFKA_HOME/bin/kafka-topics --zookeeper localhost:2181 --
list"
alias kcreate="$KAFKA_HOME/bin/kafka-topics --zookeeper localhost:2181 --
create --replication-factor 1 --partitions 1 --topic"
alias kcreate5="$KAFKA_HOME/bin/kafka-topics --zookeeper localhost:2181 --
create --replication-factor 1 --partitions 5 --topic"
alias kcreate10="$KAFKA_HOME/bin/kafka-topics --zookeeper localhost:2181 -
-create --replication-factor 1 --partitions 10 --topic"
alias kcreate15="$KAFKA_HOME/bin/kafka-topics --zookeeper localhost:2181 -
-create --replication-factor 1 --partitions 15 --topic"
alias kcreate20="$KAFKA_HOME/bin/kafka-topics --zookeeper localhost:2181 -
-create --replication-factor 1 --partitions 20 --topic"
alias kdelete="$KAFKA_HOME/bin/kafka-topics --zookeeper localhost:2181 --
delete --topic"
alias kdescribe="$KAFKA_HOME/bin/kafka-topics --zookeeper localhost:2181 -
-describe --topic"


# run
nohup $KAFKA_HOME/bin/zookeeper-server-start
$KAFKA_HOME/etc/kafka/zookeeper.properties > /home/logs/zookeeper.log &
nohup $KAFKA_HOME/bin/kafka-server-start
$KAFKA_HOME/etc/kafka/server.properties > /home/logs/kafka.log &
nohup /home/programs/stream-reactor/bin/start-connect.sh >
/home/logs/stream-reactor.log &
```

5.3. InfluxDB

InfluxDB [70], zaman serilerine odaklanmış bir veritabanıdır. Zamana sıkı bağlı olan metriklerin saklanması ve üzerinde yoğun okuma/yazma işlemlerinin yapılabilmesine olanak sağlamaktadır. Veritabanında her kayıt bir zaman dizini ile tutulmaktadır. Bu sayede zaman tabanlı sorgular hızla çalıştırılabilmekte ve veri analizleri hızla yapılabilmektedir. Aynı zamanda, belirli bir süreyi geçen verilerin otomatik olarak silinmesi ve verilerin küçültülmesi (down-sampling) de desteklenmektedir.

InfluxDB, çalışma kapsamında, sürekli akan-verinin zaman içerisindeki davranışının kayıt altına alınabilmesi için kullanılmıştır. Aşağıda InfluxDB kurulum adımları verilmiştir.

```
wget https://dl.influxdata.com/influxdb/releases/influxdb-1.2.0.x86_64.rpm
yum -y localinstall influxdb-1.2.0.x86_64.rpm
vim /etc/influxdb/influxdb.conf
* admin/enabled = true
* admin/bind-address = ":8083"

# profile
echo “export INFLUXDB_CONFIG_PATH=/etc/influxdb/influxdb.conf” >>
~/.bash_profile
echo “export INFLUXDB_ADMIN_ENABLED=true” >> ~/.bash_profile

# start service
service influxdb start
influxdb
> CREATE DATABASE "dkg"
```

5.4. Grafana

Grafana [71], zaman serilerine bağlı kayıt edilmiş verilerin görselleştirilmesini sağlayan bir uygulamadır. Sistemlerin zamana bağlı olarak gözlemlenebilmesi ve analiz edilmesi için kullanılır. Grafana ile sisteme özel gösterge tabloları oluşturulabilir, grafikler çizdirilebilir ve hatta alarmlar yaratılarak belirli koşullarda uyarılar verilebilir. Çalışma kapsamında sistemin çalışmasının ve zaman içerisindeki davranışının gözlemlenebilmesi için kullanılmıştır. Aşağıda Grafana kurulum adımları verilmiştir.

```
wget https://grafanarel.s3.amazonaws.com/builds/grafana-4.1.2-
1486989747.x86_64.rpm
yum install -y grafana-4.1.2-1486989747.x86_64.rpm
service grafana-server start
```

5.5. Java, Storm, Kafka

Uygulama Java tabanlı geliştirilmiş olup JDK 1.8 kullanılmıştır. Storm 0.9.7, Kafka ise 0.10.2.0 sürümü ile kullanılmıştır. Uygulamaya ait bütün kaynak kodları GitHub üzerinden paylaşılmıştır [72].

5.6. Uygulamanın Paketlenmesi

Deneyin yapılabilmesi için, aşağıdaki komutlarla uygulama kaynak kodunun indirilmesi ve derleme yapılarak çalıştırılacak hale getirilmesi gerekmektedir.

```
mkdir /home/programs/dkg && cd /home/programs/dkg
git clone https://github.com/odalabasmaz/DynamicKeyGrouping.git
cd DynamicKeyGrouping
mvn clean package
> target/dkg-wd.jar
```

6. DENEY ve UYGULAMALAR

6.1. Deney Ortamı

Deneyin yapıldığı sunucuya ait özellikler şu şekildedir:

Fujitsu Siemens Primergy TX200S6

- 2 adet Intel Xeon E5620
- 48 GB RAM
- DVD-RW supermulti slimline SATA
- MountingKit DVD(sl) + LSP/LSD + SAS LFF HDD
- RAID Ctrl SAS 6G 5/6 512MB (D2616)
- RAID Contr BBU Upgrade for RAID 5/6 V55
- 3 adet HD SAS 3G 146GB 15K HOT PLUG 3.5" EP
- 2 adet Power Supply Module 800W HE (hot plug)

6.2. Veri Kümesi

Deney kapsamında 5 gerçek veri kümesi ile 12 adet yapay veri kümesi kullanılmıştır. *Gerçek veri* kümelerini **twitter** ve **wikipedia** içerikleri oluşturmaktadır. Bu veriler;

- **twitter-election:** ABD 2016 yılı başkanlık seçimlerinin yapıldığı günlerde atılan tweet'leri içermektedir [73]. Veri kümesinde 11.860.067 tweet bulunurken, toplamda 5.316.612 adet hashtag içermektedir.
    - Ör. "'hashtags":[{"text":"VOTE"}]'"
- **twitter-ticker:** Borsadaki firmaların kısaltma adlarına ticker denmektedir. [26] çalışmasında da kullanılan bu veri kümesinde ticker verilerini içeren tweet'ler incelenmiştir. Veri kümesi 1.465.450 adet ticker içermektedir.
    - Ör. "1383264000 AAPL"
- **wikipedia-clickstream:** Wikipedia sayfalarına, hangi kaynaktan (diğer sayfalar) ve kaç kere erişim sağlandığını içeren veri kümeleridir [74]. Veri kümesinde 25.755.132 adet clickstream kaydı bulunurken, toplamda 8.012.700.871 adet anahtar değerine sahip açılan sayfa bilgisi bulunmaktadır.
    - Ör. "Carbon_monoxide_poisoning Aristotle link 15"

- **wikipedia-pageviews:** Wikipedia sayfalarının görüntülenme bilgilerini içermektedir [26]. Her sayfanın hangi dil seçeneğiyle kaç kez görüntülendiği verisi tutulmaktadır. Veri kümesi 21.713.921 adet pageview kaydı içermektedir.
    - Ör. “tr George_Lucas 5 0”
- **wikipedia-pageviews-by-lang:** Wikipedia sayfalarının görüntülenme bilgilerini içermektedir [75]. wikipedia-pageviews’de olduğu gibi hangi dil seçeneğiyle kaç kez görüntülendiği verisi tutulmaktadır, ancak, bu veri kümesinde farklı olarak dil seçeneğine odaklanarak çarpık veri elde edilmeye çalışılmıştır. Çünkü wikipedia sayfaları milyonlarca kayıt içerse de içerikler birkaç dil üzerinde yoğunlaşmaktadır. Veri kümesinde 160.392.361 adet kayıt bulunurken, toplamda anahtar değeri taşıyan 588.214.391 adet sayfa bilgisi bulunmaktadır.
    - Ör. “tr Lale_Devri 2 0”

Yapay veriler ise dünya üzerindeki 204 **ülkenin isimlerinden** oluşmaktadır. Yük dağıtım yöntemlerinin farklı çarpıklık oranlarında nasıl davrandığını gözlemlemek ve çıkarsamalarda bulunabilmek adına, farklı oranlarda çarpıklık içeren veri kümeleri üretilmiştir. Bu kapsamda ülkeler arasından “turkey” kaydı seçilmiş ve sırasıyla %0, %10, %20, %30, %40, %50, %60, %70, %80, %90, %100 oranlarında “turkey” kaydını içeren yapay veri kümeleri üretilmiştir. Ek olarak, DKG yönteminin sadece genişlemediği, veriye bağlı olarak, dinamik bir şekilde küçüldüğünü de gösterebilmek adına, ilk yarısı %80 yoğunluğa sahip olup, ikinci yarısı dengeli dağıtılmış bir veri kümesi de üretilmiştir. Bu kapsamda 12 adet veri kümesi üretilmiştir ve bütün veri kümeleri 10.000.000 kayıt içermektedir.

Çalışma kapsamında yararlanılan gerçek ve yapay bütün veri kümelerine ait bilgiler Çizelge 2’de yer almaktadır. Bu çizelgede her veri kümesinin türü, içerdiği kayıt sayısı ve toplam anahtar sayısı gibi bilgilerin yanı sıra, çarpıklık oranları da verilmektedir.

Çizelge 2. Veri Kümelerinin Çarpıklık Oranları

Veri Kümesi	Veri İçeriği	Veri Türü	Toplam Kayıt Sayısı	Toplam Anahtar Sayısı	Çarpıklık Oranı
twitter-election	tweet hashtags	Gerçek	11,860,067	5,316,612	%68
twitter-ticker	tweet tickers	Gerçek	1,465,450	1,465,450	%10
wikipedia-clickstream	clickstream	Gerçek	25,755,132	8,012,700,871	%10
wikipedia-pageviews	pageviews	Gerçek	21,713,921	21,713,921	%9
wikipedia-pageviews-by-lang	pageviews by lang	Gerçek	160,392,361	588,214,391	%27
country-skew-r0	ülke isimleri	Sentetik	10,000,000	10,000,000	%0
country-skew-r10	ülke isimleri	Sentetik	10,000,000	10,000,000	%10
country-skew-r20	ülke isimleri	Sentetik	10,000,000	10,000,000	%20
country-skew-r30	ülke isimleri	Sentetik	10,000,000	10,000,000	%30
country-skew-r40	ülke isimleri	Sentetik	10,000,000	10,000,000	%40
country-skew-r50	ülke isimleri	Sentetik	10,000,000	10,000,000	%50
country-skew-r60	ülke isimleri	Sentetik	10,000,000	10,000,000	%60
country-skew-r70	ülke isimleri	Sentetik	10,000,000	10,000,000	%70
country-skew-r80	ülke isimleri	Sentetik	10,000,000	10,000,000	%80
country-skew-r90	ülke isimleri	Sentetik	10,000,000	10,000,000	%90
country-skew-r100	ülke isimleri	Sentetik	10,000,000	10,000,000	%100
country-half-skew-r80	ülke isimleri	Sentetik	10,000,000	10,000,000	%40

6.3. Verilerin Kafka'ya Aktarılması

6.3.1. Kafka Konularının Oluşturulması

İşlenecek veriler Kafka üzerinden okunacağı için, elimizde bulunan ham veri kümelerinin, uygulamamızın işleyebileceği bir düzende Kafka'ya aktarılması gerekmektedir. Bu nedenle öncelikle Kafka üzerinde her veri kümesi için ayrı bir konu oluşturmamız gerekmektedir.

Uygulamalar Kafka konularına abone olarak verileri okuyabilmektedirler. Bu okuma işlemi bölüntüler üzerinden yapılmakta ve bir uygulama bir bölüntüden okuma yapabilmektedir [8]. Bu nedenle, Kafka konusunun bölüntü sayısının uygulamamızdaki Kaynak Birim sayısına denk olmasına dikkat etmemiz gerekmektedir.

Aşağıdaki kod kısmında 5 bölüntülü bir konu oluşturma örneği verilmiştir. Deney kapsamında 10, 15 ve 20 bölüntülü konular da benzer şekilde oluşturulmuştur.

```
# kafka konusu oluşturma
kcreate5 twitter-election-5
kcreate5 twitter-ticker-5
kcreate5 wikipedia-clickstream-5
kcreate5 wikipedia-pageviews-5
kcreate5 wikipedia-pageviews-by-lang-5
```

6.3.2. Veri Kümesi Şablonları

Her veri kümesi farklı tür veriler içermekle birlikte farklı yapılara da sahiptir. Bu nedenle, her bir veri kümesi için ayrı bir şablon belirlenmiş olup, ham veriler, veri içeriğine göre farklı şekillerde okunmuş olup, Kafka'ya ise yeni bir şablon ile JSON formatında aktarılmıştır. Aşağıda her veri türü için şablon detayları verilmiştir.

6.3.2.1. twitter-election

twitter-election veri kümesinde atılan her tweet'in detaylı içerik bilgisi JSON formatında bulunmaktadır. Aşağıda tweet verisinin önemli alanları örnek olarak verilmiştir. Bütün veri kümesi bu şablona göre okunup, **text**, **hashtags** ve **timestamp** alanları Kafka'ya yeni bir veri yapısı oluşturularak JSON formatında aktarılmıştır.

```
{
   “created_at”: “string”,
   “id”: “long”,
   “text”: “string”,
   “entities”: {
      “hashtags”: [{"text": "string"}]
   },
   "lang": "string",
   "timestamp_ms": "string",
   …
}
```

```
{
   “text”: “string”,
   “hashtags”: [{"text": "string"}],
   "timestamp": "long"
}
```

twitter-election verisi Kafka'ya aktarılırken sadece text, hashtags ve timestamp alanları seçilmiştir. Bu sayede Kafka disk kullanımı için, toplamda 55GB büyüklüğüne sahip veri kümesinde %99'a yakın bir kazanım sağlanmıştır.

6.3.2.2. twitter-ticker

twitter-ticker veri kümesi açık metin formatında, her satırda bir zaman damgasının yanında hisse kod bilgisiyle birlikte oluşmaktadır. Bütün alanlar olduğu gibi alınıp

```
"timestamp" "ticker"
```

```
{
    "ticker": "string",
    "timestamp": "long"
}
```

6.3.2.3. wikipedia-clickstream

wikipedia-clickstream veri kümesi açık metin formatında, her satırda bir clickstream bilgisi yer alacak şekilde oluşmaktadır. Veride, önceki sayfa, yeni açılan sayfa, bağlantı türü ve toplamda kaç kez bu sayfanın bu yolla açıldığı bilgileri yer almaktadır. Bütün alanlar olduğu gibi alınıp Kafka'ya JSON formatında aktarılmıştır.

```
"prev page" "curr page" "type" "count"
```

```
{
    "prev": "string",
    "curr": "string",
    "type": "string",
    "n": "long"
}
```

6.3.2.4. wikipedia-pageviews

wikipedia-pageviews veri kümesi açık metin formatında, her satırda bir zaman damgası ile görüntülenen sayfanın URL bağlantısından oluşmaktadır. Bütün alanlar olduğu gibi alınıp Kafka'ya JSON formatında aktarılmıştır.

"timestamp" "page"	{ "page": "string", "timestamp": "long" }

6.3.2.5. wikipedia-pageviews-by-lang

wikipedia-pageviews-by-lang veri kümesi açık metin formatında, her satırda görüntülenen sayfanın dili, sayfanın adı, görüntülenme sayısı ve ek bir bilgiden oluşmaktadır. Bütün alanlar olduğu gibi alınıp Kafka'ya JSON formatında aktarılmıştır.

"lang" "page" "n" "m"	{ "lang": "string", "page": "string", "n": "long", "m": "long" }

6.3.2.6. country

country veri kümesinde açık metin formatında, her satırda bir ülkenin ismi geçmektedir. İsimler olduğu gibi alınıp, Kafka'ya JSON formatında aktarılmıştır.

"country"	{ "country": "string" }

6.4. Uygulama

Bu kesimde, uygulamanın mimarisi hakkında bilgiler verilecek olup, sonrasında uygulamanın çalıştırılmasına ilişkin bilgilendirmeler yapılacaktır.

6.4.1. Uygulama Mimarisi

Storm ilingesinde veri giriş noktaları Kaynak Birimleridir. Kafka'da tutulan veriler Kaynak Birim aracılığıyla ilingeye giriş yaparlar. Veri okuma sırasında maksimum performans ile çalışabilmek için Kaynak Birim sayısının Kafka bölüntü sayısına denk olmasına dikkat edilmesi gerekmektedir. Uygulamada 5, 10, 15 ve 20 adet Kaynak Birim ile testler koşulmuştur.

Uyguama mimarisi Şekil 19'da verilmiştir. Bu mimaride, Kaynak Birimlerinden giriş yapan veriler Ayrıştırıcı Birimlere SG yöntemi ile rastgele dağıtılırlar. Uygulamada Ayrıştırıcı sayısı 10 olarak belirlenmiştir ve Ayrıştırıcıların görevi farklı içeriklere sahip verileri ilingenin çalışabileceği tek tip veri yapısına çevirmektir. Bu nedenle her veri kümesi için farklı Ayrıştırıcı Birimleri tanımlanmıştır. İlingemiz, veri kümelerindeki anahtar kelimeleri saptayıp bunların sayısını çıkartmak için tasarlandığından, Ayrıştırıcı Birimler *<Anahtar Kelime, Sayı, Zaman Damgası>* veri yapısı ile çıktı (çok-öğeli) üretmektedir. Anahtar kelimelerin saptanması şöyle olmaktadır:

- **twitter-ticker:** tweet'lerin içerisinde yer alan ticker sembolleri anahtar kelime olarak alınmıştır. Sayı = 1 olarak tanımlanmış olup zaman damgası verinin içerisinde yer aldığı şekliyle kullanılmıştır.
- **twitter-election:** tweet'lerin içerisindeki hashtag'ler anahtar kelime olarak alınmıştır. Sayı = 1 olarak tanımlanmış olup zaman damgası verinin içerisinde yer aldığı şekliyle kullanılmıştır. Ayrıca, twitter verisi okunurken, verinin %11 olan doğal çarpıklık oranını arttırmak adına ilişkili olan hashtag'lerin birleştirilmesi sağlanmıştır. Seçimlerle ilgili aynı veya benzer anlamları içeren sözcükler filtrelenmiş olup tek bir sözcükle değiştirilmiş ve seçimle ilgili atılmış olan bütün tweet'lerin ortaklaştırılması sağlanmıştır. Bu kapsamda büyük küçük harf duyarlılığı yapmaksızın, içerisinde *"hilary, hillary, clinton, withher, wither, voteher, lockherup, donald, trump, makeamericagreatagain, americafirst, vote, voting, poll, election, eleicoes, eleccion, electon, electoral, abdseçimleri,*

president, debate, democrat, republican, america, us, usa" geçen hashtag'ler "ElectionDay" ile değiştirilmiş ve bu sayede %68'lik bir çarpıklık oranı yakalanmıştır.

- **wikipedia-clickstream:** clickstream verisindeki açılan sayfa adı bilgisi anahtar kelime olarak alınmıştır. Sayı değeri verinin içerisinde yer alan tıklanma sayısından alınmış olup, verinin içerisinde zaman damgası yer almadığından, işlemin yapıldığı zamanın bilgisi, zaman damgası olarak kullanılmıştır.
- **wikipedia-pageviews:** pageviews verisindeki açılan sayfanın URL bilgisi alınıp, hepsinde aynı olan "http://en.wikipedia.org/wiki/" öneki çıkartılarak anahtar kelime olarak alınmıştır. Sayı = 1 olarak tanımlanmış olup zaman damgası verinin içerisinde yer aldığı şekliyle kullanılmıştır.
- **wikipedia-pageviews-by-lang:** pageviews verisindeki açılan sayfaların dil bilgisi anahtar kelime olarak alınmıştır. Bu kapsamda, wikipedia-pageviews ile farklı olması ve çarpık veri kümesi oluşturabilmesi adına açılan sayfa bilgisi dikkate alınmamış sadece sayfaların dil bilgisi dikkate alınmıştır. Sayı değeri verinin içerisinde yer alan tıklanma sayısından alınmış olup, verinin içerisinde zaman damgası yer almadığından, işlemin yapıldığı zamanın bilgisi, zaman damgası olarak kullanılmıştır.
- **country:** country veri kümesinde yer alan ülke isimleri anahtar kelime olarak alınmıştır. Sayı = 1 olarak tanımlanmış olup verinin içerisinde zaman damgası yer almadığından, işlemin yapıldığı zamanın bilgisi, zaman damgası olarak kullanılmıştır.

Ayrıştırıcı Birimden çıkan veriler İşçi Birimlere belirli yöntemlere göre gönderilmektedir. Bu yöntemler Kesim 2.3.8'de belirtilen ve bu çalışma kapsamında Kesim 4'de önerilen yöntemlerdir. Gönderilecek İşçi Birimin seçilmesinde ise bu yöntemler (Karışık Gruplama hariç) belirlenen anahtar kelimeleri kullanarak tercih yapmaktadırlar.

Ayrıştırıcı Birimden çıkan veriler, ilingenin kurulum amacına göre belirli görevleri yapmak ve verileri işlenmek üzere İşçi Birimlere gönderilirler. İşçi Birimler veriyi işleyecek ve çıktı üretecek asıl iş birimleridir. Çalışma kapsamında veriyi işleyen İşçi Birimler, gelen her veri için 1 milisaniye bekleyerek gerçek zamanlı işleri taklit etmiştir.

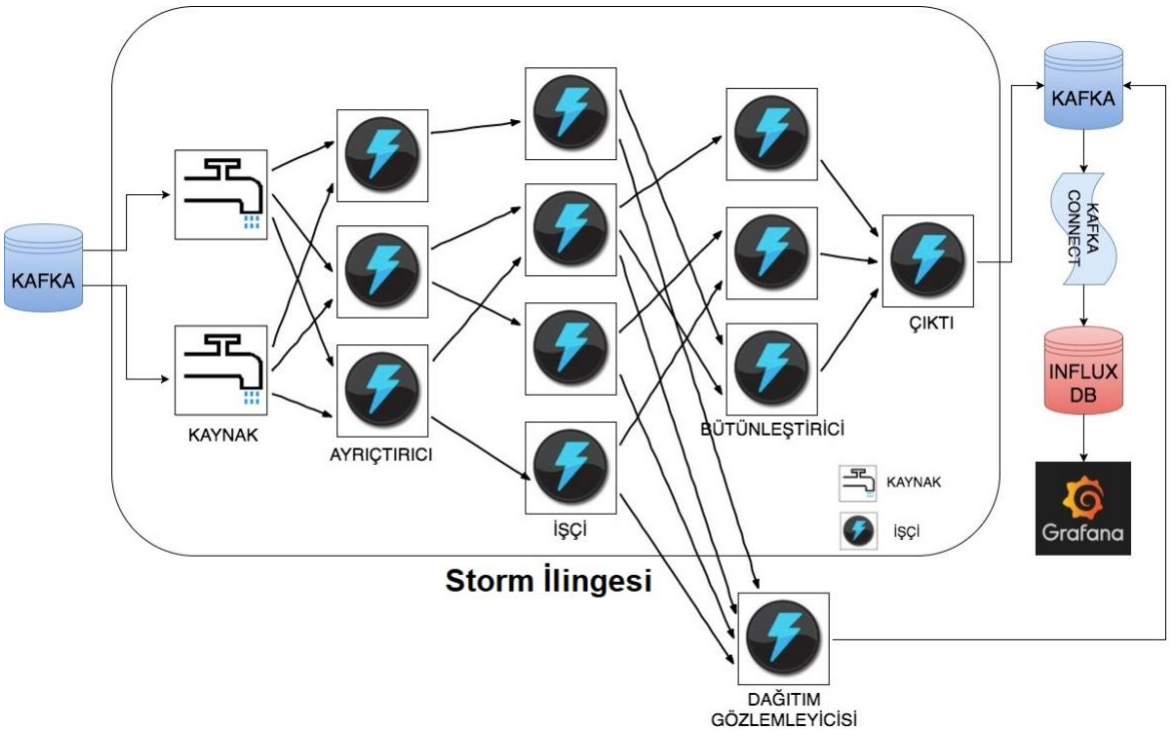
İşçi Birimler işledikleri verileri hemen bir sonraki aşamaya göndermezler, bunun yerine belirli bir süre işlenen verileri biriktirip, belirli aralıklarla toptan gönderirler. Bu sayede ağ trafiğini yoğunlaştırmayarak, sistemlerin daha performanslı ve sağlıklı çalışması sağlanabilmektedir. Tabii ki bu sürenin çok fazla olması da sistemin performansını olumsuz etkileyecektir. Çünkü daha uzun süre beklenmesi, daha çok verinin toplanması ve daha büyük verinin ağa bir anda bırakılması anlamına gelecektir. Bu da ağ yoğunluğu yaratacaktır. Bunun yanı sıra, gerçek zamanlı veri işlendiği için, anlık olayların sonucu daha geç görülebilecektir. Bu çalışma kapsamında İşçi Birimler 15 saniyelik aralıklarla işledikleri verileri bir sonraki aşamaya geçirmektedir.

İşçi Birimlerde işlenen veriler, sonrasında Bütünleştirici Birime KG yöntemi kullanılarak aktarılırlar. Bu sayede, aynı anahtar değerine sahip olan verilerin aynı makineye aktarılması sağlanır. Aynı anahtar değerine sahip olan veriler birleştirilir ve tek bir sonuç haline getirilirler. İşçi Birimde olduğu gibi, burada da 60 saniye boyunca verilerin toplanma ve birleştirilme işlemi devam ederken sonrasında sonuç çıktısı üretebilmek adına Çıktı Birimine gönderilirler.

Çalışma kapsamında önerilen yöntemin ve diğer yöntemlerin performansının gözlemlenebilmesi adına, ilingeye fazladan bir birim daha eklenmiştir. Gözlemci Birim adı verilen bu birim, Bütünleştirici Birim gibi, İşçi Birimde işlenerek çıkan verileri kendi üzerinde toplar ve detaylı bir analiz yaparak sonuçları Çıktı Birimine gönderir. Bu sayede, yöntemlerin çalışma süreleri, üretkenlikleri, birim zamanda yaptıkları işler, işlenen veri miktarı, işlerin İşçi Birimler üzerindeki dağılımı vb. birçok metrik toplanmaktadır. Bu metrikler, yöntemlerin karşılaştırılması ve yorumlanabilmesi için bize ışık tutmaktadır.

Çıktı Birimine gelen veriler ise, hiçbir değişiklik yapılmadan Kafka'ya gönderilmektedir. Bu sayede verilerin güvenliği ve kalıcılığı sağlanmış olmaktadır. Kafka'ya yazılan veriler ise anlık olarak KafkaConnect ile InfluxDB'ye aktarılmaktadır. Grafana ise InfluxDB'ye yazılan verileri okuyarak, grafikler, tablolar ve göstergeler sağlamakta ve böylece sistemi anlık takip edebileceğimiz bir ortam sağlamaktadır.

Şekil 19'da, çalışma kapsamında gerçekleştirilen uygulamanın ve ilingenin görseli verilmiştir.



Şekil 19. Deney ortamı için oluşturulan uygulama ilingesi

6.4.2. Uygulamanın Çalıştırılması

Uygulamanın çalıştırılabilmesi için, Kesim 5.5'deki yönergeleri uygulayarak *dkg-wd.jar* paketinin oluşturulması gerekmektedir. Daha sonra bu paket kullanılarak uygulama farklı parametreler ile çalıştırılabilir. Bu parametreler, hangi veri kümesinin kullanılacağını, hangi yöntemin işletileceğini, sistemde kaç tane Kaynak Birim ve kaç tane İşçi Birim bulunacağını belirlememizi sağlamaktadır. Bu parametreler ve alabileceği değerler şu şekildedir:

- **STORM_MODE:** İlingenin yerel makinede veya kümelenmiş makine grubunda çalıştırılacağını belirler. Alabileceği değerler:
  - [LOCAL|CLUSTER]
- **DATA_SET:** Hangi veri kümesinin kullanılacağını belirler. Alabileceği değerler:
  - [TWITTER_TICKER|WIKIPEDIA_PAGEVIEWS|COUNTRY_SKEW_R0|COUNTRY_SKEW_R10|COUNTRY_SKEW_R20|COUNTRY_SKEW_R30|COUNTRY_SKEW_R40|COUNTRY_SKEW_R50|COUNTRY_SKE

W_R60|COUNTRY_SKEW_R70|COUNTRY_SKEW_R80|COUNTRY_SKEW_R90|COUNTRY_SKEW_R100|COUNTRY_HALF_SKEW_R80|TWITTER_ELECTION|WIKIPEDIA_CLICKSTREAM|WIKIPEDIA_PAGEVIEWS_BY_LANG]

- **ALGO:** Uygulamanın hangi yöntem ile çalıştırılacağını belirler. Bu yöntemler bu çalışmanın kapsamını oluşturan yük dağıtım yöntemleridir. Alabileceği değerler:
  - [SHUFFLE|KEY|PARTIAL_KEY|DYNAMIC_KEY]
- **KAFKA_TOPIC:** Uygulamanın veriyi hangi Kafka konularından okuyacağını belirler. Veriler hangi konuya yazıldıysa onun isminin verilmesi gerekmektedir. Alabileceği değerde kısıtlama yoktur. Örneğin, twitter-ticker-5, 5 bölüntüden oluşmuş ve Kaynak sayısı 5 olan bir deney için oluşturulmuş Kafka konusudur.
- **SPOUT_COUNT:** Kafka konularına bağlanacak ve verileri okuyacak olan ilingedeki Kaynak Birim sayısını belirler. Alabileceği değerler tam sayılar olmakla birlikte çalışma kapsamında kullanılan değerler şunlardır:
  - [5|10|15|20]
- **WORKER_COUNT:** Uygulamada çalışacak olan İşçi Birimlerinin sayısını belirler. Alabileceği değerler tam sayılar olmakla birlikte çalışma kapsamında kullanılan değerler şunlardır:
  - [5|10|20|50|100]
- **SPEED:** Uygulamanın veri işleme hızını belirler, aslında taklit eder. Daha önce de belirtildiği üzere, veri işleme 1ms'lik duraksamalarla taklit edilmiştir. SPEED parametresi ile bu duraksamalar değiştirilerek veri işleme hızının artırılması taklit edilir. Bu çalışma kapsamında hep x1 kullanılmış olup, daha büyük veri kümeleri ile çalışırken, daha kısa zamanda sonuç alabilmek için x10 ve x100 değerleri kullanılabilir. Alabileceği değerler, başında "x" olmak kaydıyla bütün tam sayılardır.
  - [x1|x10|x100]

Uygulama, her veri kümesi, Kaynak Birim sayısı, İşçi Birim sayısı ve her yöntem için ayrı ayrı çalışabilecek şekilde yapılandırılmıştır. Arzu edilirse, uygulamanın çalışma günlüğü ayrı bir dosyaya yönlendirilerek takip edilebilir. Uygulamanın çalıştırılmasına ait şablon ve örnek çalıştırma şekli aşağıda verilmiştir.

```
nohup java $JAVA_OPTS -jar dkg-wd.jar <STORM_MODE> <DATA_SET> <ALGO>
<KAFKA_TOPIC> <SPOUT_COUNT> <WORKER_COUNT> <SPEED> > /home/logs/dkg/log.out &

nohup java $JAVA_OPTS -jar dkg-wd.jar LOCAL TWITTER_TICKER DYNAMIC_KEY
twitter-ticker-5 5 10 x1 > /home/logs/dkg/twitter-ticker-5-s5-w10-dkg.out &
```

6.5. Deney Çıktılarının Yorumlanması

Deney kapsamında gerçek veya yapay içerikten oluşan birçok veri kümesi kullanılmıştır. Bu veriler farklı Kaynak Birim ve İşçi Birim sayılarına sahip ilingelerde işlenmiş ve yöntemlerin çeşitli veri kümeleri, Kaynak Birim ve İşçi Birim sayılarına göre nasıl çalıştığı gözlemlenmiştir. Farklı parametrelerle gerçekleştirilmiş olan deneyler, seçilen parametrelere göre gruplandırılıp sonuçları aşağıda detaylandırılmıştır.

Grafikler performans karşılaştırması yapılabilmesi adına maksimum değerleri üzerinden düzgelenerek çizdirilmiştir. Bu kapsamda en yüksek değer 100'e sabitlenecek şekilde bütün değerler orantılı olarak güncellenmiştir.

Gecikme, yöntemlerin çalışma sürelerini, Dağıtım Maliyeti bir anahtar kelimenin kaç farklı makineye dağıtıldığını, Standart Sapma, makineler arası dağıtımın ne kadar dengeli yapıldığını ve Üretkenlik de birim zamanda ne kadar çıktı üretildiğini göstermektedir. Yöntemlerden, verimli olabilmeleri adına, Gecikme, Dağıtım Maliyeti ve Standart Sapma değerlerinin düşük, Üretkenlik değerinin ise yüksek olması beklenmektedir. Özellikle, Dağıtım Maliyeti ve Standart Sapma değerlerinin düşük olması amaçlanmaktadır. Bu sayede, sistem hem bütün makineleri verimli kullanabilmekte hem de makinelere dağıtılan verilerin tekrar bir araya toplanma maliyeti en aza indirgenebilmektedir. Grafikler bu beklentiler üzerinden yorumlanacak olup

yöntemler kısaca SG (Karışık Gruplama), KG (Anahtar Gruplama), PKG (Parçalı Anahtar Gruplama) ve DKG (Dinamik Anahtar Gruplama) olarak anılacaktır.

Yöntemlerin yapısı gereği, SG yönteminde dağıtım maliyetinin yüksek, standart sapmanın düşük olması beklenirken, KG yönteminde dağıtım maliyetinin düşük, standart sapmanın yüksek olması beklenmektedir. PKG yönteminde ise, KG yöntemine nazaran yüksek dağıtım maliyeti ve düşük standart sapma beklenirken, çarpıklık oranı arttıkça değerlerin KG yöntemine yakınlaşması beklenmektedir. Bununla birlikte, DKG yönteminde, PKG yöntemine nazaran yüksek dağıtım maliyeti ve düşük standart sapma beklenirken, çarpıklık oranı arttıkça sistemin daha iyi değerler üretmesi beklenmektedir.

6.5.1. Gerçek Veri Kümesi ile Deney

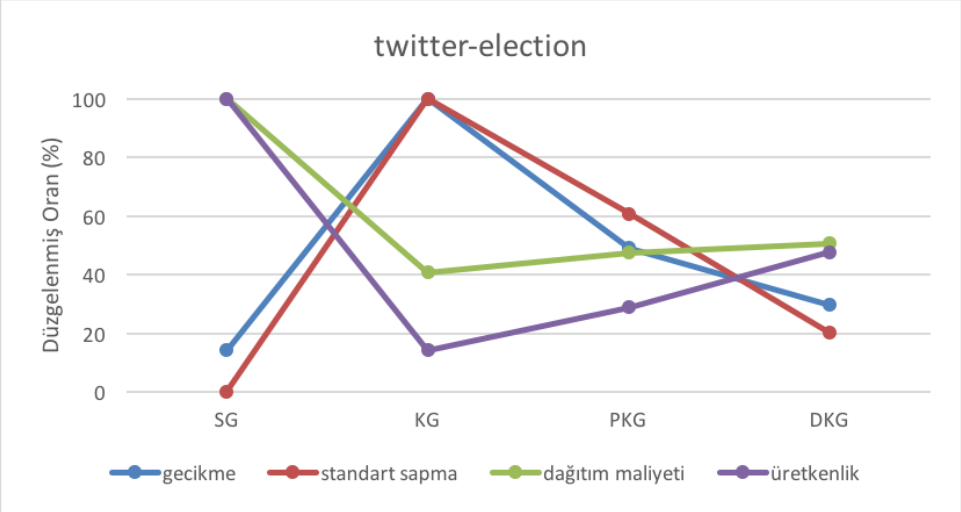
Deney, gerçek veri kümeleri üzerinde, 5 adet Kaynak Birim ve 10 adet İşçi Birim ile çalıştırılmıştır. Bu kapsamda twitter-election, twitter-ticker, wikipedia-clickstream, wikipedia-pageviews ve wikipedia-pageviews-by-lang veri kümeleri kullanılmıştır.

6.5.1.1. twitter-election veri kümesi

twitter-election veri kümesi (kayıt sayısı = 5,316,612, çarpıklık = %68) ile yapılan deneyin sonuçları Çizelge 3'te verilmiştir.

Çizelge 3. twitter-election veri kümesi ile yapılan deneyin sonuçları

YÖNTEM	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	559,329	0.0001	2.4491	9,511
KG	3,955,045	20.3202	1.0000	1,344
PKG	1,941,879	12.3513	1.1647	2,739
DKG	1,174,979	4.0972	1.2414	4,529



Şekil 20. twitter-election performans karşılaştırması

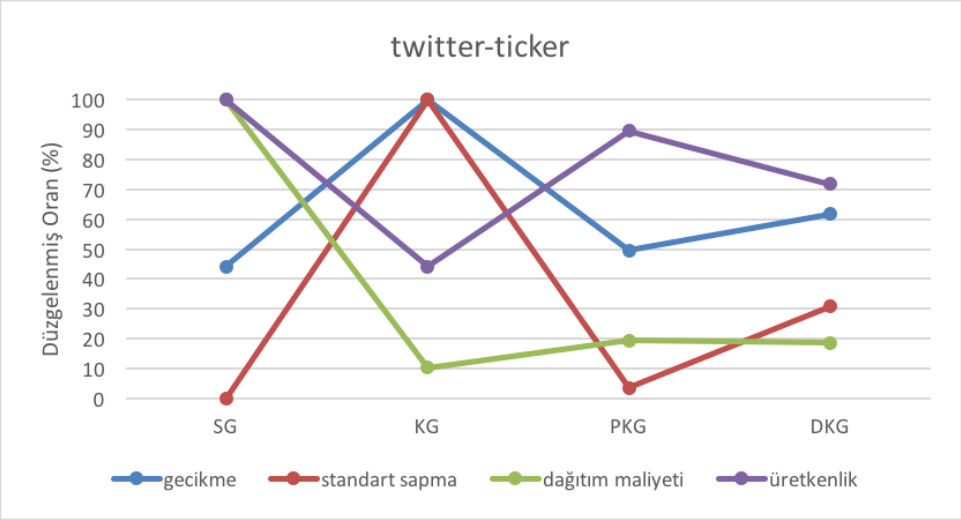
Şekil 20'de görüldüğü üzere, SG yönteminde elde edilen yüksek Dağıtım Maliyeti ile KG yönteminde elde edilen düşük Üretkenlik ile yüksek Gecikme ve Standart Sapma değerleri, sistemin performansını olumsuz etkileyerek verimsiz çalışmasını sağlamıştır. Bu kapsamda, diğer yöntemlere göre daha düşük Dağıtım Maliyeti ve Standart Sapma değerlerine sahip iki yöntemden DKG, Dağıtım Maliyetinin çok yakın olmasına rağmen, daha yüksek Üretkenlik ile daha düşük Standart Sapma ve Gecikme değerleri ile PKG yöntemine kıyasla daha başarılı olmuştur.

6.5.1.2. twitter-ticker veri kümesi

twitter-ticker veri kümesi (kayıt sayısı = 1,465,450, çarpıklık = %10) ile yapılan deneyin sonuçları Çizelge 4'te verilmiştir.

Çizelge 4. twitter-ticker veri kümesi ile yapılan deneyin sonuçları

YÖNTEM	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	137,081	0.0003	9.6444	10,697
KG	310,106	4.0913	1.0000	4,727
PKG	153,698	0.1459	1.8736	9,578
DKG	191,395	1.2591	1.7985	7,673



Şekil 21. twitter-ticker performans karşılaştırması

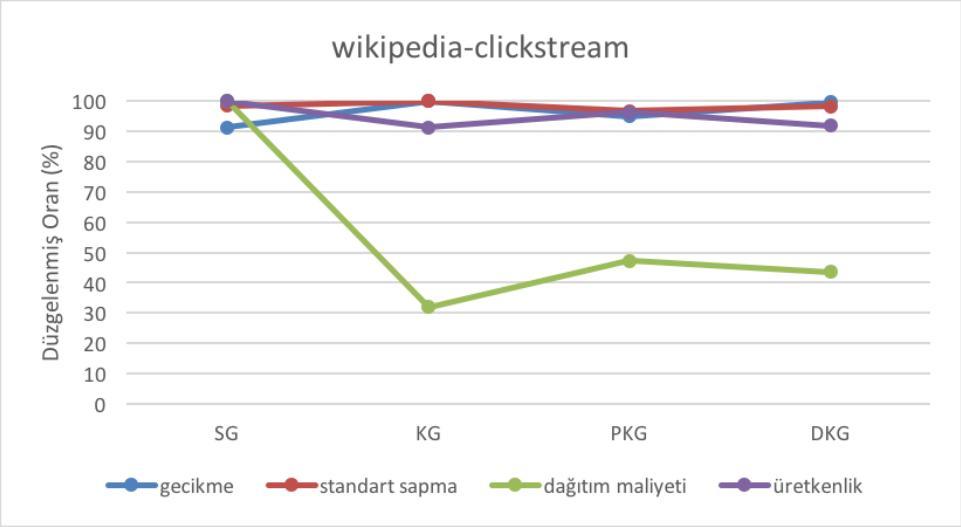
Şekil 21'de görüldüğü üzere, SG yönteminde elde edilen yüksek Dağıtım Maliyeti ile KG yönteminde elde edilen yüksek Standart Sapma değerleri, sistemin performansını olumsuz etkileyerek verimsiz çalışmasını sağlamıştır. Genele baktığımızda, Gecikme sadece KG için yüksek değere sahip olurken diğer yöntemler için çok yakın değerler almıştır. Bu kapsamda, diğer yöntemlere göre daha düşük Dağıtım Maliyeti ve Standart Sapma değerlerine sahip iki yöntemden PKG, daha yüksek Üretkenlik ile daha düşük Standart Sapma ve Gecikme değerleri ile DKG yöntemine kıyasla daha başarılı olmuştur.

6.5.1.3. wikipedia-clickstream veri kümesi

wikipedia-clickstream veri kümesi (kayıt sayısı = 8,012,700,871, çarpıklık = %10) ile yapılan deneyin sonuçları Çizelge 5'te verilmiştir.

Çizelge 5. wikipedia-clickstream veri kümesi ile yapılan deneyin sonuçları

YÖNTEM	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	2,757,313	3.0746	3.1298	2,906,312
KG	3,019,604	3.1278	1.0000	2,654,091
PKG	2,863,742	3.0238	1.4763	2,798,708
DKG	3,004,187	3.0713	1.3663	2,667,344



Şekil 22. wikipedia-clickstream performans karşılaştırması

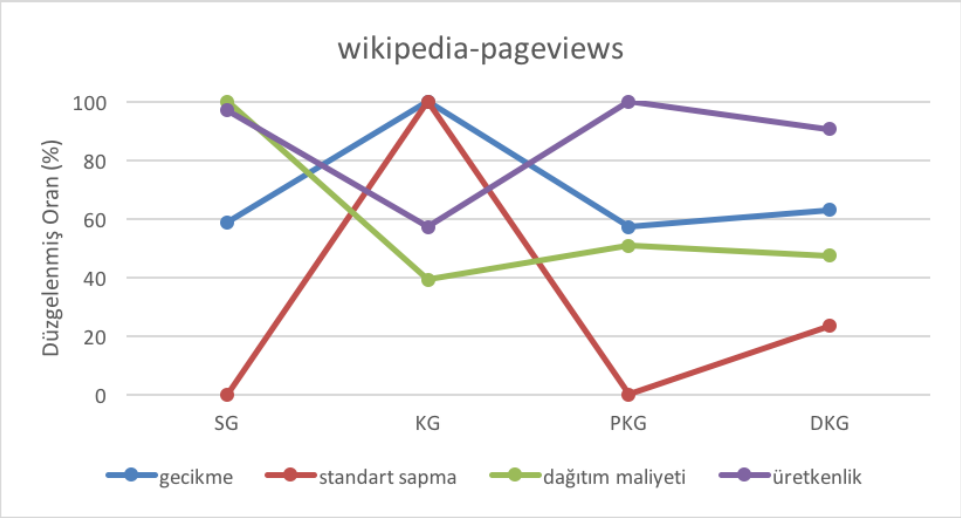
Şekil 22'de görüldüğü üzere, bütün yöntemler Gecikme, Standart Sapma ve Üretkenlik için yakın değerlere sahip olduğundan kıyaslamayı Dağıtım Maliyeti üzerinden yapabiliriz. Bu kapsamda SG yöntemi en yüksek değere sahip olurken, diğer yöntemler çok yakın değerlere sahip olmuştur. Bu veri kümesi için KG yöntemi PKG ve DKG yöntemlerine kıyasla daha düşük Dağıtım Maliyeti değerine sahip olarak daha verimli ve performanslı çalışarak daha başarılı olmuştur.

6.5.1.4. wikipedia-pageviews veri kümesi

wikipedia-pageviews veri kümesi (kayıt sayısı = 21,713,921, çarpıklık = %9) ile yapılan deneyin sonuçları Çizelge 6'da verilmiştir.

Çizelge 6. wikipedia-pageviews veri kümesi ile yapılan deneyin sonuçları

YÖNTEM	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	2,458,310	0.0000	2.5479	8,834
KG	4,177,335	2.8186	1.0000	5,198
PKG	2,388,351	0.0000	1.2948	9,093
DKG	2,631,656	0.6586	1.2089	8,253



Şekil 23. wikipedia-pageviews performans karşılaştırması

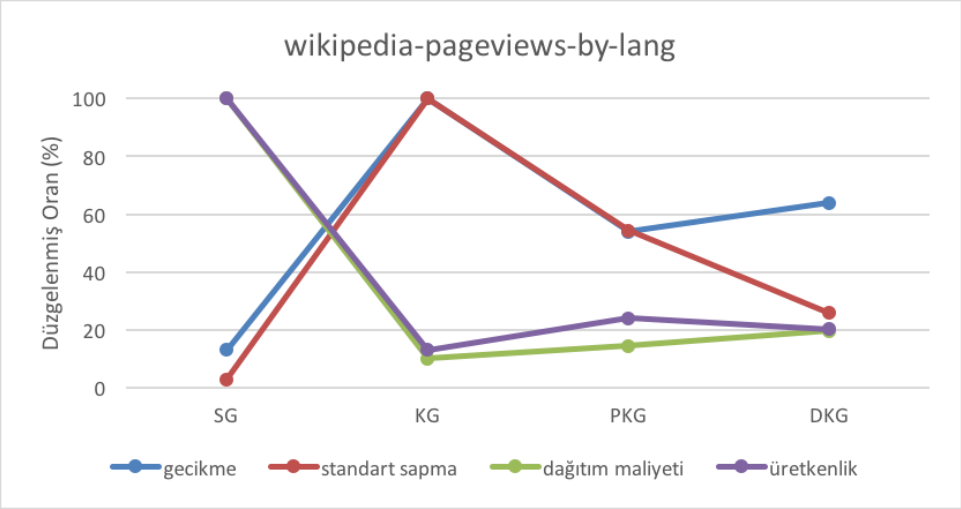
Şekil 23'de görüldüğü üzere, SG yönteminde elde edilen yüksek Dağıtım Maliyeti ile KG yönteminde elde edilen yüksek Standart Sapma ve Gecikme değerleri, sistemin performansını olumsuz etkileyerek verimsiz çalışmasını sağlamıştır. Genele baktığımızda Gecikme sadece KG yöntemi için yüksek değere sahip olurken diğer yöntemler için çok yakın değerler almıştır. Aynı şekilde Dağıtım Maliyeti de KG, PKG ve DKG yöntemleri için yakın değerler almıştır. Bu kapsamda, diğer yöntemlere göre daha düşük Dağıtım Maliyeti ve Standart Sapma değerlerine sahip iki yöntem, yakın Üretkenlik, Gecikme ve Dağıtım Maliyeti değerlerine sahip olurken, PKG yöntemi DKG yöntemine kıyasla daha düşük Standart Sapma değerine sahip olduğundan, daha verimli ve performanslı çalışarak daha başarılı olmuştur.

6.5.1.5. wikipedia-pageviews-by-lang veri kümesi

wikipedia-pageviews-by-lang veri kümesi (kayıt sayısı = 588,214,391, çarpıklık = %27) ile yapılan deneyin sonuçları Çizelge 7'de verilmiştir.

Çizelge 7. wikipedia-pageviews-by-lang veri kümesi ile yapılan deneyin sonuçları

YÖNTEM	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	16,934,614	0.4049	9.9238	34,736
KG	130,874,178	13.9257	1.0000	4,495
PKG	70,501,163	7.5734	1.4317	8,343
DKG	83,789,184	3.5902	1.9460	7,020



Şekil 24. wikipedia-pageviews-by-lang performans karşılaştırması

Şekil 24'te görüldüğü üzere, SG yönteminde elde edilen yüksek Dağıtım Maliyeti ile KG yönteminde elde edilen yüksek Standart Sapma ve Gecikme değerleri, sistemin performansını olumsuz etkileyerek verimsiz çalışmasını sağlamıştır. Üretkenliğin sadece SG yöntemi ile istenen seviyelere çıkmasına rağmen yüksek Dağıtım Maliyeti değeri yüzünden tercih edilememektedir. Üretkenlik ve Dağıtım Maliyeti değerleri diğer yöntemler için yakın değerler aldığından Standart Sapma ve Gecikme değerleri dikkate alınmıştır. Bu kapsamda KG yöntemi çok yüksek Standart Sapma değerine sahip olduğundan, daha düşük Standart Sapma değerine sahip iki yöntemden DKG, PKG ile yakın Gecikme değerine sahip olurken daha düşük Standart Sapma değerine sahip

olduğundan PKG yöntemine kıyasla daha verimli ve performanslı çalışarak daha başarılı olmuştur.

6.5.2. Yapay Veri Kümesi ile Deney

Deney, yapay veri kümeleri üzerinde, 5 adet Kaynak Birim ve 10 adet İşçi Birim ile çalıştırılmıştır. Bu kapsamda, country veri kümesi kullanılmıştır. Bu veri kümesi yapay olarak farklı çarpıklık oranlarında üretildiğinden, yöntemlerin, Kaynak Birim ve İşçi Birim sayıları ile işlenen verinin çarpıklık oranlarına nasıl tepki verdiklerinin gözlemlenebilmesi amaçlanmıştır.

6.5.2.1. country-skew veri kümesi

Deneyin ilk kısmında country veri kümesinin (kayıt sayısı = 10,000,000) %0'dan %100'e kadar olan farklı çarpıklık oranlarıyla testler koşulmuş ve performansları gözlemlenmiştir. Çizelge 8'de yapay veri kümesi ile yapılan deneylerin sonuçlarına yer verilmiştir.

Çizelge 8. Yapay veri kümesi ile yapılan deneylerin sonuçları

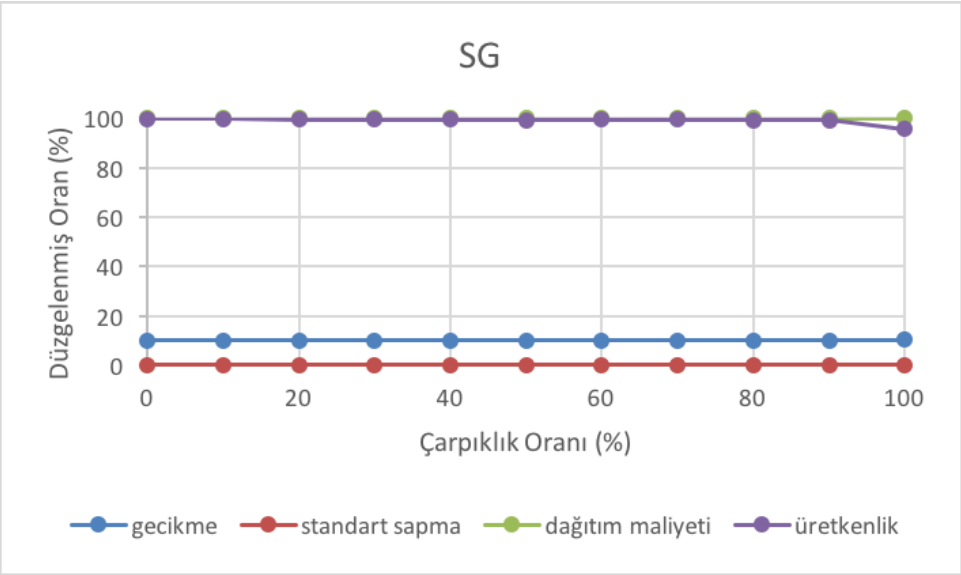
YÖNTEM	ÇARPIKLIK ORANI (%)	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	0	1,039,569	0.0000	10.0000	9,625
KG	0	1,283,021	1.7136	1.0000	7,794
PKG	0	1,046,808	0.0000	1.9265	9,560
DKG	0	1,146,397	0.6155	2.0000	8,726
SG	10	1,040,381	0.0000	10.0000	9,615
KG	10	2,015,510	3.4519	1.0000	4,963
PKG	10	1,041,919	0.0001	1.9265	9,606
DKG	10	1,252,389	1.0013	1.9510	7,987
SG	20	1,042,125	0.0000	10.0000	9,597
KG	20	2,947,223	6.2265	1.0000	3,393
PKG	20	1,250,950	0.9778	1.7598	8,000
DKG	20	1,431,979	1.8237	1.9951	6,988
SG	30	1,041,265	0.0000	10.0000	9,606
KG	30	3,887,240	9.1447	1.0000	2,573
PKG	30	1,744,071	3.3600	1.6765	5,734
DKG	30	1,611,159	2.4502	2.1520	6,207
SG	40	1,043,282	0.0000	10.0000	9,588
KG	40	4,818,445	12.1073	1.0000	2,076
PKG	40	2,248,751	5.7380	1.6225	4,448
DKG	40	1,682,432	2.9866	2.3676	5,945
SG	50	1,044,168	0.0000	10.0000	9,579
KG	50	5,752,010	15.0706	1.0000	1,739

PKG	50	2,733,002	8.1102	1.6176	3,659
DKG	50	1,836,567	3.5161	2.5098	5,447
SG	60	1,042,541	0.0000	10.0000	9,597
KG	60	6,705,053	18.0575	1.0000	1,491
PKG	60	3,228,979	10.4913	1.6127	3,098
DKG	60	1,972,658	3.8405	2.5735	5,071
SG	70	1,043,778	0.0000	10.0000	9,588
KG	70	7,639,566	21.0412	1.0000	1,309
PKG	70	3,736,871	12.8709	1.6078	2,677
DKG	70	2,065,983	4.3218	2.7549	4,843
SG	80	1,044,141	0.0000	10.0000	9,579
KG	80	8,567,183	24.0180	1.0000	1,167
PKG	80	4,223,407	15.2432	1.6078	2,368
DKG	80	2,183,984	4.6728	2.8431	4,581
SG	90	1,044,493	0.0000	10.0000	9,579
KG	90	9,499,136	27.0109	1.0000	1,053
PKG	90	4,725,702	17.6225	1.6029	2,116
DKG	90	2,308,485	4.8442	2.6765	4,333
SG	100	1,083,820	0.0000	10.0000	9,234
KG	100	10,439,235	30.0000	1.0000	958
PKG	100	5,220,439	20.0000	2.0000	1,916
DKG	100	2,391,644	1.4031	8.0000	4,182

Performanslara ait grafikler ve detaylar aşağıda verilmiştir.

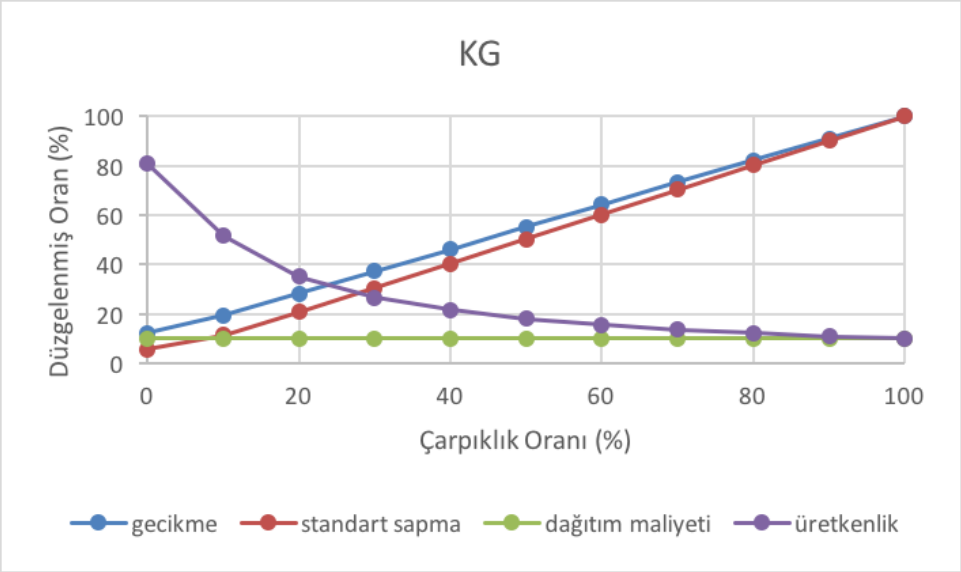
6.5.2.2. Yöntemlere Göre Deney Sonuçları

Bu kesimde yapay veri kümesi üzerinde artan çarpıklık oranlarına kıyasla yöntemlerin performansları gözlemlenmiştir.



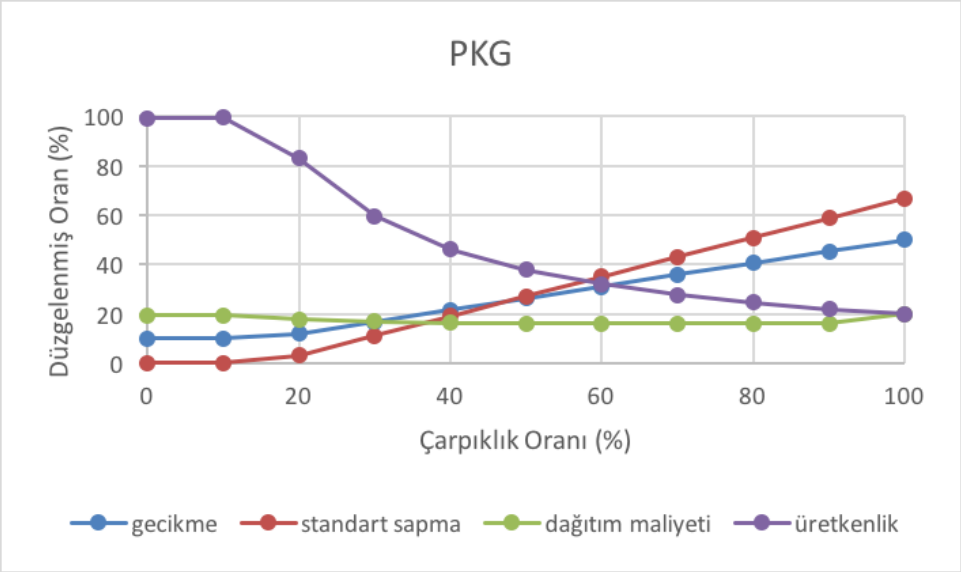
Şekil 25. SG yönteminin veri çarpıklığına göre davranışı

Şekil 25'te görüldüğü üzere, veri kümesindeki çarpıklık oranının artmasının, SG yönteminin performansını hiçbir şekilde etkilemediği gözlemlenmiştir. Veriler arası bağlantının olmadığı ve SG yönteminin kullanabildiği durumlarda, verinin içeriğinden bağımsız olarak sistemin performansı ve verimliliği öngörülebilmektedir.



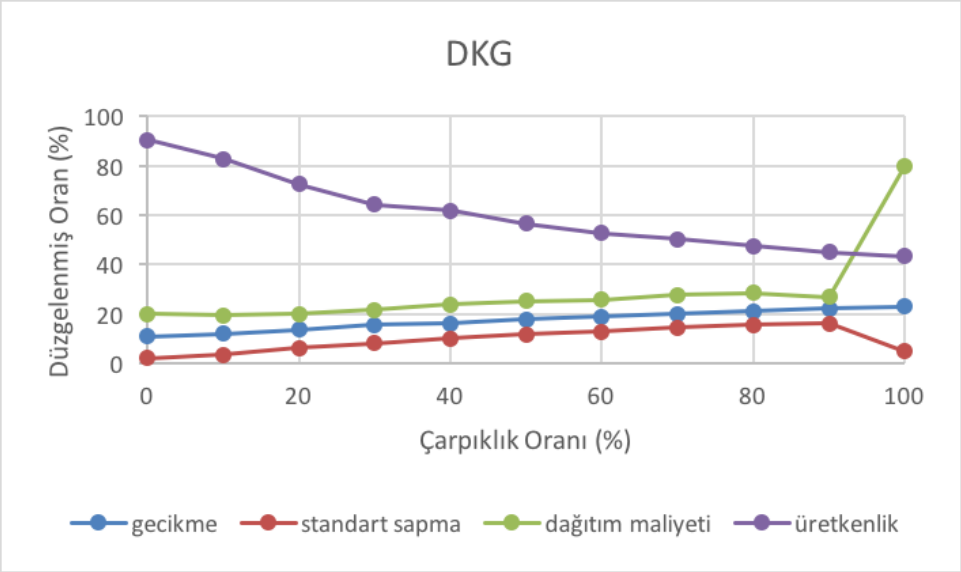
Şekil 26. KG yönteminin veri çarpıklığına göre davranışı

Şekil 26'da görüldüğü üzere, veri kümesindeki çarpıklık oranının artması, KG yönteminin Dağıtım Maliyeti değerini etkilemezken, Standart Sapma ve Gecikme değerlerini verinin çarpıklık oranıyla doğrusal olarak arttırdığı ve Üretkenlik değerini de giderek düşürdüğü gözlemlenmiştir. Bu kapsamda, veri kümesinin çarpıklık oranının özellikle %30'dan fazla olduğu durumlarda KG yönteminin performansının ve veriminin, veri kümesinin çarpıklık oranıyla doğru orantılı olarak düşeceği öngörülmektedir.



Şekil 27. PKG yönteminin veri çarpıklığına göre davranışı

Şekil 27'de görüldüğü üzere, veri kümesindeki çarpıklık oranının artması, PKG yönteminin Dağıtım Maliyeti değerini olumlu yönde çok az etkilerken, Standart Sapma ve Gecikme değerlerini özellikle %20'lik çarpıklık oranından sonra olumsuz etkilemeye başlamıştır ve çarpıklık değeri arttıkça bu değerler de artış göstermiştir. Benzer şekilde, Üretkenlik değeri de özellikle %10'luk çarpıklık oranından sonra düşüşe geçmiştir ve çarpıklık oranının artmasıyla doğru orantılı olarak bu değerde de düşüş gözlemlenmiştir. Bu kapsamda, veri kümesinin çarpıklık oranının %20'nin üzerinde olması durumunda, sistemin performansının ve verimliliğinin çarpıklık oranıyla doğru orantılı olarak, giderek azalacağı öngörülmektedir. Bununla birlikte, KG yöntemine kıyasla, PKG yöntemi veri kümesinin çarpıklık oranından daha az etkilenmekte olduğu gözlemlenmiş olup, her çarpıklık oranında KG yönteminden daha verimli ve performanslı çalışacağı öngörülmektedir.

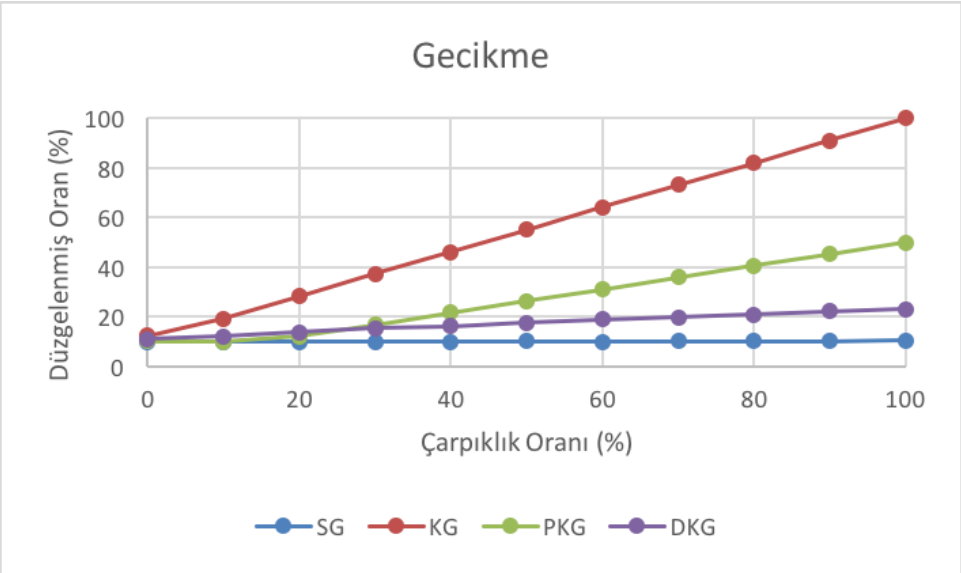


Şekil 28. DKG yönteminin veri çarpıklığına göre davranışı

Şekil 28'de görüldüğü üzere, veri kümesindeki çarpıklık oranının artması, DKG yönteminin Üretkenlik değerini çarpıklık oranıyla doğru orantılı olarak yavaşça azaltırken, Gecikme, Standart Sapma ve Dağıtım Maliyeti değerlerini ise önemsenecek ölçüde değiştirmemiştir. Bununla birlikte, özellikle yüksek çarpıklık oranlarında, DKG yöntemi KG ve PKG yöntemine kıyasla daha yüksek Üretkenlik değerlerine sahip olurken, Gecikme, Standart Sapma ve Dağıtım Maliyeti için de oldukça düşük değerlere sahip olarak daha verimli ve performanslı çalışacağını göstermiştir. DKG yöntemi, Gecikme, Standart Sapma ve Dağıtım Maliyeti değerlerini sabit tutarak ve Üretkenlik değerini diğer yöntemlere göre çok az düşürerek, çarpıklık oranının artmasından en az etkilenen yöntem olmaktadır. Bu kapsamda, özellikle çarpık veriler ile çalışacak sistemlerin tahmin edilebilir ve öngörülebilir performanslar göstererek her daim verimli çalışabilmesini sağlayacağı öngörülmektedir.

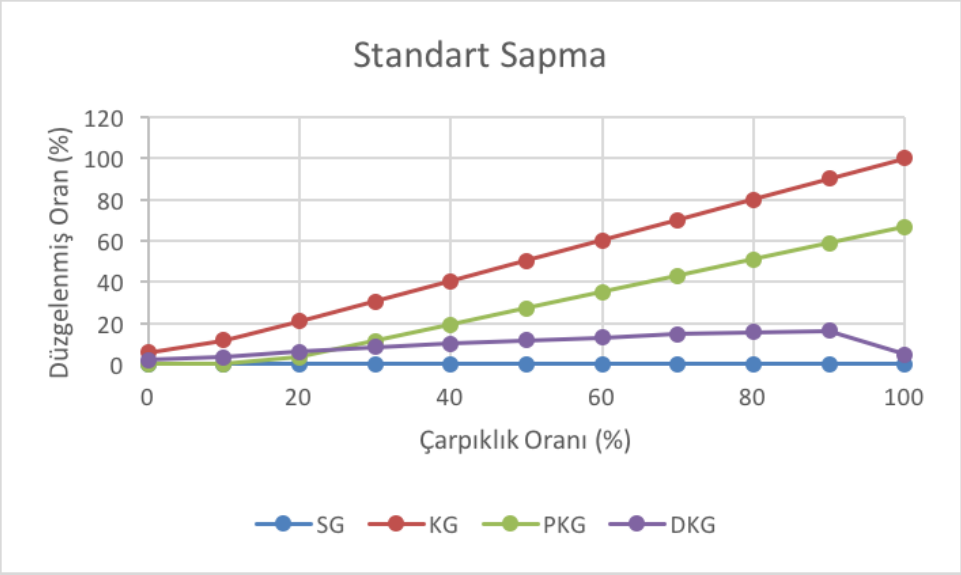
6.5.2.3. Metriklere Göre Deney Sonuçları

Bu kesimde yapay veri kümesi üzerinde yöntemlerin metriklere göre performansları gözlemlenmiştir.



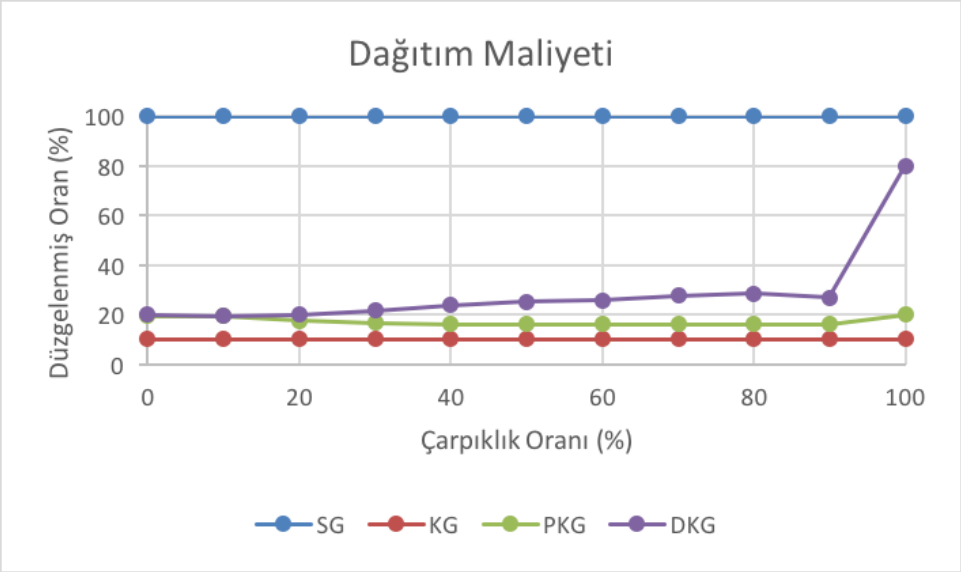
Şekil 29. Gecikme değerinin çarpıklık oranına göre değişimi

Şekil 29'da görüldüğü üzere, veri kümesindeki çarpıklık oranının artması, Gecikme değerleri bakımından, SG yöntemini hiç etkilemezken en çok KG yöntemini etkilemiştir. SG yönteminin seçilemediği durumlardaysa, çarpıklıktan en az etkilenen yöntem DKG olmuştur. PKG yönteminde %30'luk çarpıklık oranından sonra doğrusal artış gözlemlenmiş olup, DKG yöntemi hemen hemen hiç artış göstermeyerek, verinin çarpıklığından etkilenmemiştir. Bu kapsamda DKG yönteminin, çarpık veri kümelerinde KG ve PKG yöntemine kıyasla verinin çarpıklığından daha az etkilenerek daha düşük Gecikme değerlerine sahip olacağı ve böylece daha yüksek performans ve verimlilik göstereceği öngörülmektedir.



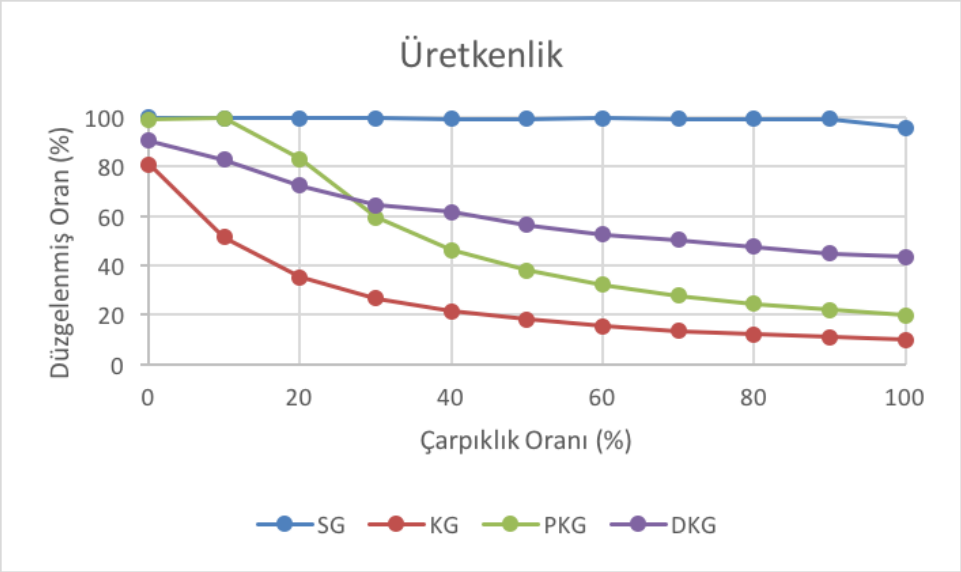
Şekil 30. Standart Sapma değerinin çarpıklık oranına göre değişimi

Şekil 30'da görüldüğü üzere, veri kümesindeki çarpıklık oranının artması, Standart Sapma değerleri bakımından, SG yöntemini hiç etkilemezken en çok KG ve PKG yöntemlerini etkilemiştir. Özellikle %20'lik çarpıklık oranından sonra her iyi yöntemde de Standart Sapma değerleri çarpıklık oranıyla doğru orantılı olarak artış gözlemlenmiştir. Bunun yanı sıra DKG yöntemi çarpıklık oranının artmasından hemen hemen hiç etkilenmemiştir ve sabit düzeyde kalmıştır. Bu kapsamda DKG yönteminin, çarpık veri kümelerinde KG ve PKG yöntemine kıyasla verinin çarpıklığından daha az etkilenerek daha düşük Standart Sapma değerlerine sahip olacağı ve böylece daha yüksek performans ve verimlilik göstereceği öngörülmektedir.



Şekil 31. Dağıtım Maliyeti değerinin çarpıklık oranına göre değişimi

Şekil 31'de görüldüğü üzere, veri kümesindeki çarpıklık oranının artması, Dağıtım Maliyeti değerleri bakımından, SG, KG ve PKG yöntemlerini hiç etkilemezken DKG yöntemini sadece %90'lık çarpıklık oranından sonra etkilemiştir. SG yönteminde her veri her makineye dağıtıldığından, en yüksek Dağıtım Maliyeti değeri, dolayısıyla da en verimsiz performans elde edilmiştir. Öte yandan KG yöntemi sadece bir, PKG yöntemi ise en fazla iki makineye dağıtım yaptığından bu yöntemlerin Dağıtım Maliyeti değerleri sabit kalmış ve SG yöntemine göre daha verimli sonuçlar üretmişlerdir. DKG yöntemi ise birden fazla makineye dağıtım yapabiliyor olmasına karşılık, çarpıklık oranının artmasıyla Dağıtım Maliyeti değerinde çok büyük artışlar gözlemlenmemiştir. KG ve PKG yöntemlerine kıyasla biraz daha yüksek Dağıtım Maliyeti değerine sahip olmasına karşın, özellikle yüksek çarpıklığa sahip veri kümelerinde Standart Sapma, Gecikme ve Üretkenlik değerlerini de hesaba kattığımızda verinin çarpıklığından daha az etkilenerek daha yüksek performans ve verimlilik göstereceği öngörülmektedir.



Şekil 32. Üretkenlik değerinin çarpıklık oranına göre değişimi

Şekil 32'de görüldüğü üzere, veri kümesindeki çarpıklık oranının artması, Üretkenlik değerleri bakımından, SG yöntemini hiç etkilememiştir. SG yöntemi ile en yüksek Üretkenlik değerleri elde edilmiş ve yüksek performans gözlemlenmiştir. Ancak, veriler arası bağlantının olduğu ve verinin içeriğine bakan yöntemlerin tercih edilmesi gerektiği durumlarda ise KG yöntemi verinin çarpıklığının artmasıyla beraber en düşük Üretkenlik değerlerine sahip olmuştur. KG yöntemi %10'luk çarpıklık değerinde dahi düşük performans göstermeye başlamıştır. Öte yandan, PKG ve DKG yöntemleri birbirlerine çok yakın Üretkenlik değerlerine sahip olmuştur. %30'luk çarpıklık oranlarına kadar PKG yöntemi daha yüksek Üretkenlik değerine sahipken, %40 ve daha yüksek çarpıklık oranlarında DKG yöntemi daha yüksek Üretkenlik değerlerine sahip olmuştur. Bu kapsamda, özellikle %40 üzeri çarpıklığa sahip durumlu veri kümelerinde DKG yönteminin diğer yöntemlere kıyasla daha yüksek Üretkenlik değerlerine sahip olarak daha yüksek performans ve verimlilik göstereceği öngörülmektedir.

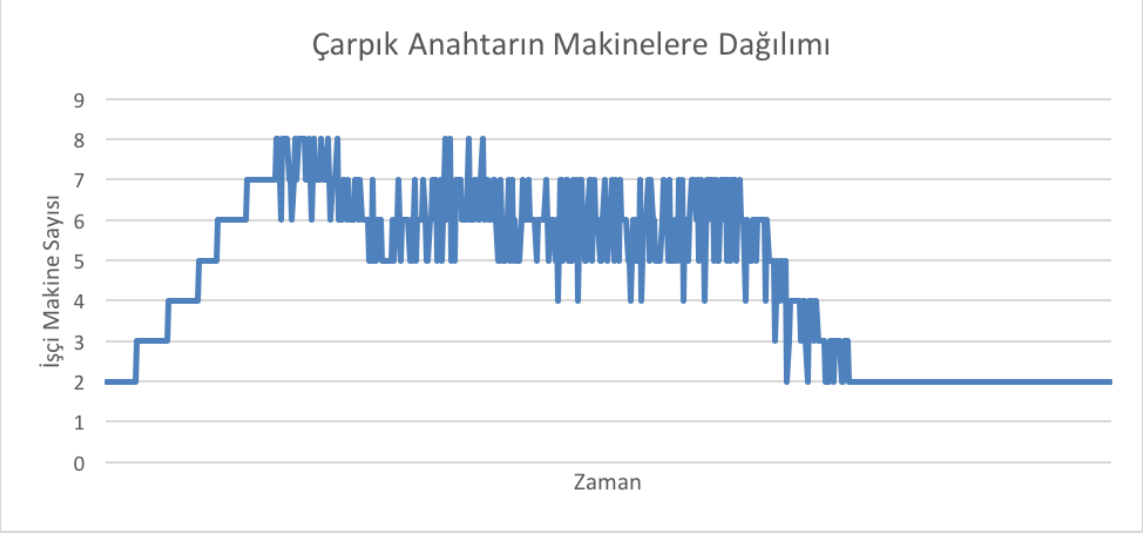
6.5.2.4. DKG Algoritmasının Değişen Veriye Adaptasyonu

DKG yönteminin değişen veri çarpıklarına uyumunu sınamak için country-skew-r80-half-skew (kayıt sayısı = 10,000,000) veri kümesi kullanılmıştır. Veri kümesine ait deney sonuçları Çizelge 9'da verilmiştir.

Çizelge 9. Veri çarpıklığı değişen country veri kümesi ile yapılan deneyin sonuçları

YÖNTEM	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	1,045,333	0.0000	10.0000	9,569
KG	4,920,867	12.1005	1.0000	2,033
PKG	2,732,391	5.7335	1.6176	3,660
DKG	2,068,104	1.6208	2.6569	4,836

Bu deney ile DKG yönteminin verinin içeriğine göre dağıtılacak makine sayısını dinamik olarak değiştiği gösterilmek istenmiştir. Kullanılan veri kümesinin yarısı %80'lik çarpıklığa sahipken, diğer yarısı ise homojen dağıtılmış verilerden oluşmaktadır. Böylece, verinin ilk yarısı işlenirken sistem daha fazla makine kullanacak olup, ikinci yarısı işlenirken kullanılan makine sayısını azaltarak tekrar dengeyi kuracaktır. Bu kapsamda gerçekleştirilen deneye ait çarpık verinin makinelere dağılımının zaman içerisindeki değişimi Şekil.33'de gösterilmiştir.



Şekil 33. Çarpık verinin makinelere dağılımının zaman içerisindeki değişimi

Şekil 33’de görüldüğü üzere, başlangıçta 2 makineye dağıtılan yük, verinin çarpıklığıyla kademeli olarak 8 makineye kadar çıkmış ve 6 ile 8 makine arasında dalgalanmıştır. Sonrasında verinin çarpıklığının azalmasıyla tekrar kademeli azalış göstererek 2 makineye kadar inmiştir.

6.5.3. İşçi Birim Sayısının Değişiminin Etkisi

Deney, gerçek ve yapay veri kümeleri üzerinde 5 adet Kaynak Birim karşılığında, 10, 50 ve 100 adet İşçi Birim ile çalıştırılmıştır. Bu kapsamda twitter-election, wikipedia-pageviews-by-lang ve country veri kümeleri kullanılmış olup, yöntemlerin artan İşçi Birim sayıları karşısındaki performansları gözlemlenmiştir.

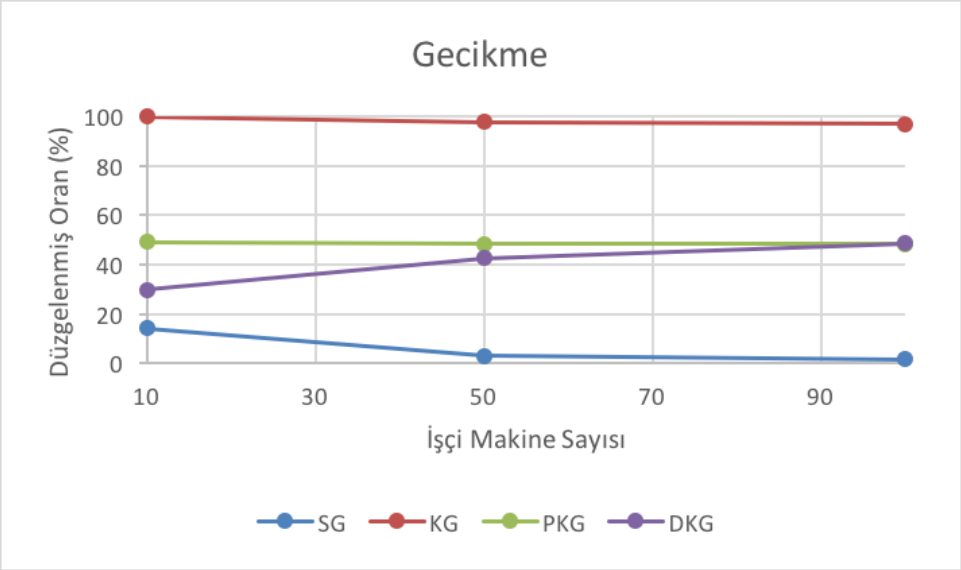
6.5.3.1. twitter-election veri kümesi

twitter-election veri kümesi (toplam kayıt = 5,316,612, çarpıklık = %68) ile yapılan deneyin sonuçları Çizelge 10'da verilmiştir.

Çizelge 10. twitter-election veri kümesi ile yapılan deneyin sonuçları

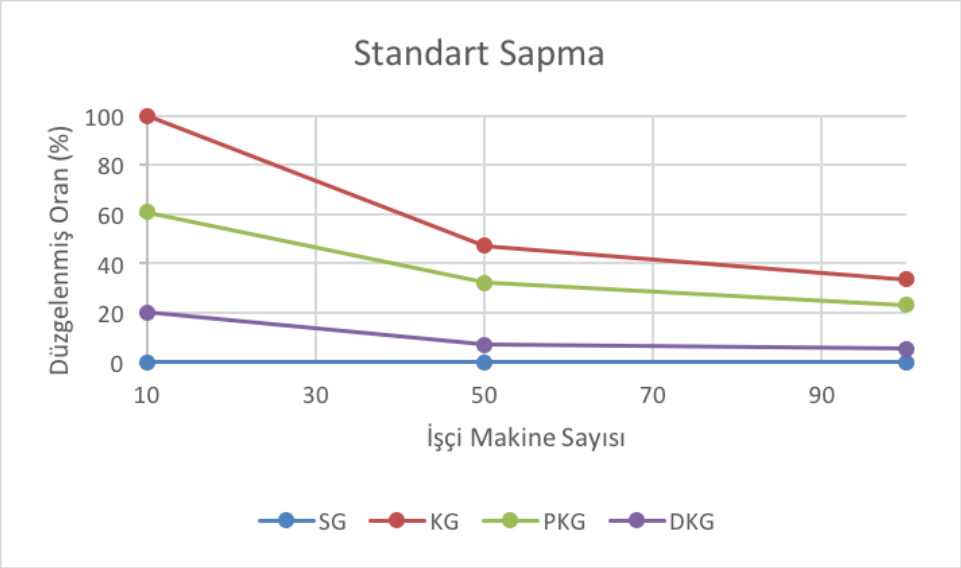
YÖNTEM	İŞÇİ MAKİNE SAYISI	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	10	559,329	0.0001	2.4491	9,511
KG	10	3,955,045	20.3202	1.0000	1,344
PKG	10	1,941,879	12.3513	1.1647	2,739
DKG	10	1,174,979	4.0972	1.2414	4,529
SG	50	117,399	0.0001	3.9349	45,441
KG	50	3,867,216	9.6129	1.0000	1,375
PKG	50	1,910,991	6.5617	1.2271	2,784
DKG	50	1,682,705	1.4353	1.1106	3,161
SG	100	60,181	0.0001	4.6933	88,610
KG	100	3,829,748	6.8101	1.0000	1,389
PKG	100	1,907,242	4.7344	1.1944	2,788
DKG	100	1,922,068	1.1045	1.0510	2,766

%68 çarpıklığa sahip twitter-election veri kümesinde, İşçi Birim sayısındaki artışın Standart Sapma değerlerinde azalışa sebebiyet verirken, Gecikme, Dağıtım Maliyeti ve Üretkenlik değerlerinde ise değişikliğe sebebiyet vermediği gözlemlenmiştir. Gözlemlere ait grafikler ve detaylı açıklamalar ise aşağıda verilmiştir.



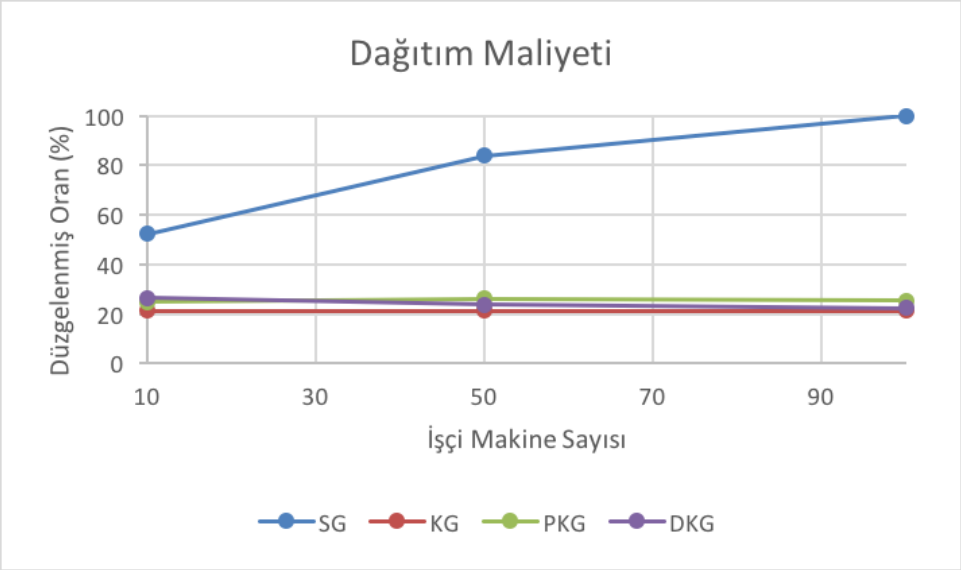
Şekil 34. Gecikme değerlerinin İşçi Birim sayılarına göre değişimi

Şekil 34'de görüldüğü üzere, İşçi Birim sayısının artması, Gecikme değerleri kapsamında, KG ve PKG yöntemlerinde bir değişikliğe yol açmamıştır. SG yönteminde İşçi Birim sayısının artmasıyla Gecikme değerinin düştüğü gözlemlenmiştir. Öte yandan, DKG yönteminde ise İşçi Birim sayısı arttıkça Gecikme değerlerinin arttığı gözlemlenmiştir. Yani DKG yönteminde İşçi Birim sayısının artması çalışma süresini azaltmanın aksine süreyi uzatmıştır ancak yine de her zaman PKG yönteminden daha düşük Gecikme değerlerine sahip olmuştur. Bu kapsamda, DKG yönteminin tercih edilebilirliği, Standart Sapma, Dağıtım Maliyeti ve Üretkenlik değerleri de göz önüne alınarak değerlendirilmelidir.



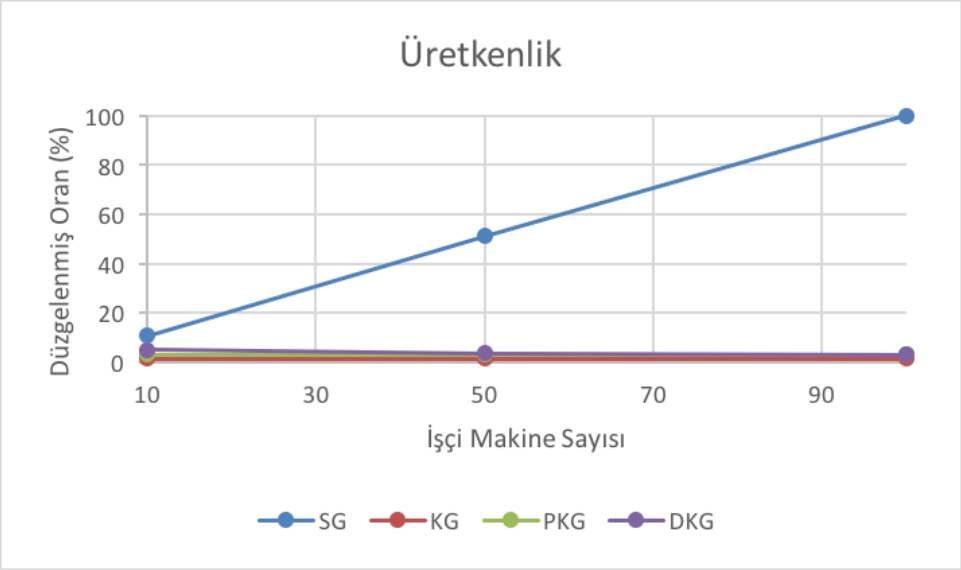
Şekil 35. Standart Sapma değerlerinin İşçi Birim sayılarına göre değişimi

Şekil 35'de görüldüğü üzere, İşçi Birim sayısının artması, Standart Sapma değerleri kapsamında, SG yönteminde bir değişikliğe yol açmazken, diğer yöntemlerde ise düşüşe yol açmıştır. Özellikle İşçi Birim sayısının 10'dan 50'ye çıkartılmasıyla Standart Sapma değerlerinde ciddi düşüşler gözlemlenmişken, 100'e çıkartılmasıyla ise 50'ye çıkartılmasına kıyasla daha az bir düşüş gözlemlenmiştir. Bu kapsamda, İşçi Birim sayısının artmasıyla Standart Sapma değerlerinin daha da düşeceği ve böylece yükün makinelere daha dengeli dağıtılarak daha yüksek performans ve verimlilik elde edileceği öngörülmektedir.



Şekil 36. Dağıtım Maliyeti değerlerinin İşçi Birim sayılarına göre değişimi

Şekil 36'da görüldüğü üzere, İşçi Birim sayısının artması, Dağıtım Maliyeti değerleri kapsamında, KG, PKG ve DKG yöntemlerinde bir değişikliğe yol açmazken, SG yönteminde ise artışa sebebiyet vermiştir. KG, PKG ve DKG yöntemleri benzer değerler üretirken, SG yönteminde İşçi Birim sayısının artmasıyla Dağıtım Maliyeti değerinin de arttığı gözlemlenmiştir. Dağıtım Maliyeti değerinin artmasıyla durumlu verilerin toplanmasının maliyeti artmaktadır ve dolayısıyla bu durum performansı da etkilemektedir. Bu kapsamda, KG, PKG ve DKG yöntemlerinin tercih edilmesi durumunda İşçi Birim sayısının arttırılmasına bağlı olarak Dağıtım Maliyetinin değişmesi beklenmezken, performans ve verimliliğinin azalmayacağı öngörülmektedir. Bununla birlikte, SG yönteminde ise, İşçi Birim sayısının artışıyla doğru orantılı olarak Dağıtım Maliyetinin artması beklendiğinden performans kayıpları ve verimliliğin azalacağı öngörülmektedir.



Şekil 37. Üretkenlik değerlerinin İşçi Birim sayılarına göre değişimi

Şekil 37'de görüldüğü üzere, İşçi Birim sayısının artması, Üretkenlik değerleri kapsamında, KG, PKG ve DKG yöntemlerinde bir değişikliğe yol açmazken, SG yönteminde ise artışa sebebiyet vermiştir. KG, PKG ve DKG yöntemleri benzer değerler üretirken, SG yönteminde İşçi Birim sayısının artmasıyla Üretkenlik değerinin de arttığı gözlemlenmiştir. Bu kapsamda, KG, PKG ve DKG yöntemlerinin tercih edilmesi durumunda İşçi Birim sayısının arttırılmasına bağlı olarak Üretkenlik değerinin değişmesi beklenmezken, performans ve verimliliğinin azalmayacağı öngörülmektedir. Bununla birlikte, durumlu verilerin hedeflenmediği SG yönteminde ise, İşçi Birim sayısının artışıyla doğru orantılı olarak Üretkenlik değerinin artması beklendiğinden performans artışı ve verimliliğin artacağı öngörülmektedir.

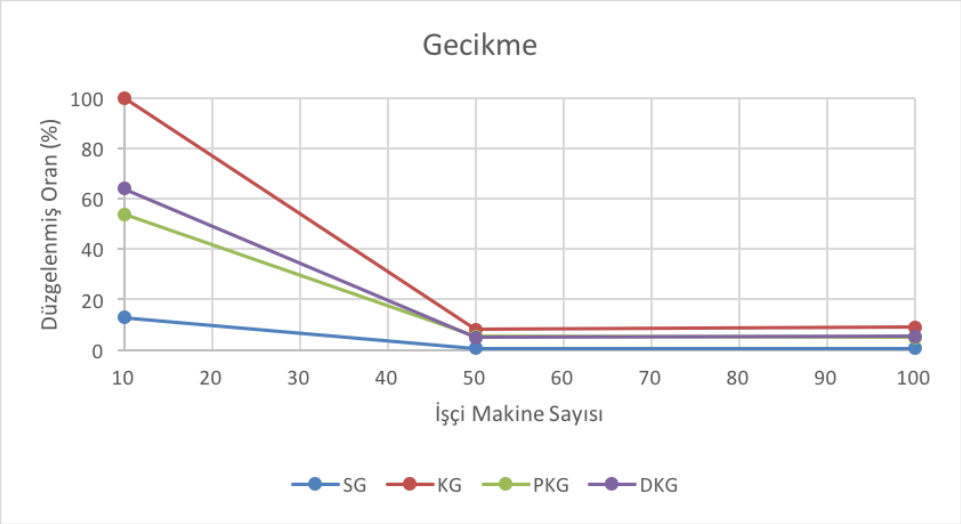
6.5.3.2. wikipedia-pageviews-by-lang veri kümesi

wikipedia-pageviews-by-lang veri kümesi (kayıt sayısı = 588,214,391, çarpıklık = %27) ile yapılan deneyin sonuçları Çizelge 11'de verilmiştir.

Çizelge 11. wikipedia-pageviews-by-lang veri kümesi ile yapılan deneyin sonuçları

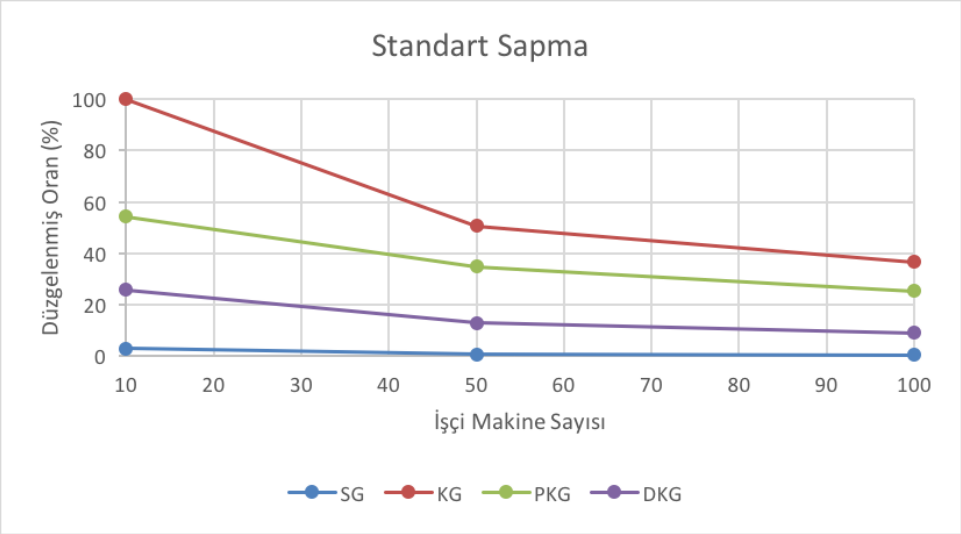
YÖNTEM	İŞÇİ MAKİNE SAYISI	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	10	16,934,614	0.4049	9.9238	34,736
KG	10	130,874,178	13.9257	1.0000	4,495
PKG	10	70,501,163	7.5734	1.4317	8,343
DKG	10	83,789,184	3.5902	1.9460	7,020
SG	50	901,217	0.1045	49.0317	652,846
KG	50	10,598,228	7.0698	1.0000	55,502
PKG	50	7,200,444	4.8461	1.4286	81,696
DKG	50	6,455,563	1.7886	1.7079	91,125
SG	100	962,166	0.0789	96.4317	611,449
KG	100	11,719,794	5.0877	1.0000	50,193
PKG	100	6,617,305	3.5343	1.4698	88,894
DKG	100	7,183,959	1.2644	1.5683	81,890

%27 çarpıklığa sahip wikipedia-pageviews-by-lang veri kümesinde, İşçi Birim sayısındaki artışın Gecikme ve Standart Sapma değerlerinde azalışa sebebiyet verirken, Dağıtım Maliyeti ve Üretkenlik değerlerinde ise değişikliğe sebebiyet vermediği gözlemlenmiştir. Gözlemlere ait grafikler ve detaylı açıklamalar ise aşağıda verilmiştir.



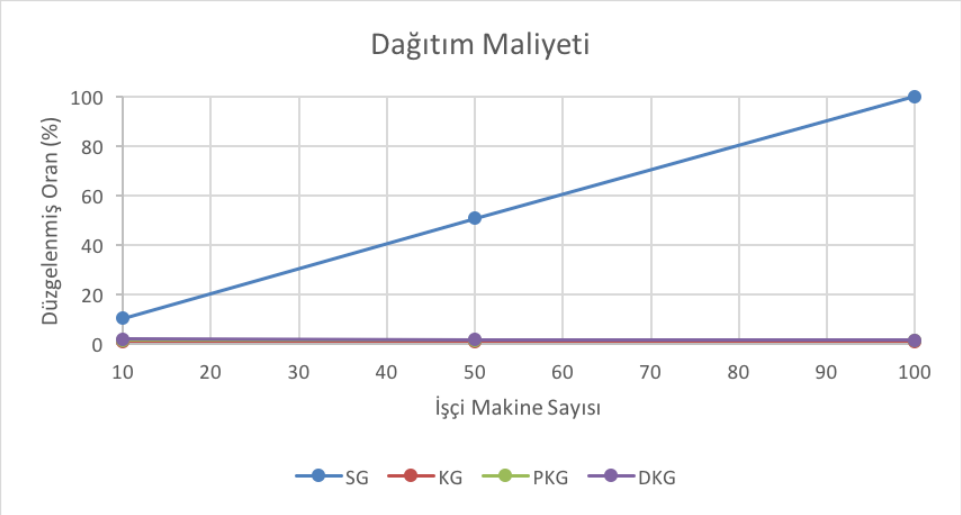
Şekil 38. Gecikme değerlerinin İşçi Birim sayılarına göre değişimi

Şekil 38'de görüldüğü üzere, İşçi Birim sayısının artması, Gecikme değerleri kapsamında, bütün yöntemlerde azalmaya yol açmamıştır. 10 adet İşçi Birim bulunduğunda en düşük Gecikme SG yöntemine ve en yüksek Gecikme KG yöntemine aitken, PKG ve DKG yöntemleriyse yakın Gecikme değerlerine sahip olmuştur. Öte yandan, İşçi Birim sayısının artmasıyla bütün yöntemlerin Gecikme değerleri oldukça düşmüş ve birbirlerine çok yakın değerler almıştır. Bu kapsamda, İşçi Birim sayısının yüksek olduğu durumlarda bütün yöntemlerin benzer performans sergileyeceği, düşük İşçi Birim sayısı ile çalışıldığı durumda ise ilişkili verilerin kullanıldığı durumda DKG ve PKG yöntemlerinin en iyi performansı ve verimliliği sergileyeceği öngörülmektedir. Verilerin ilişkisiz olduğu koşullarda ise SG yöntemiyle, İşçi Birim sayısından bağımsız olarak her zaman en iyi performans ve yüksek verimliliğin elde edileceği öngörülmektedir.



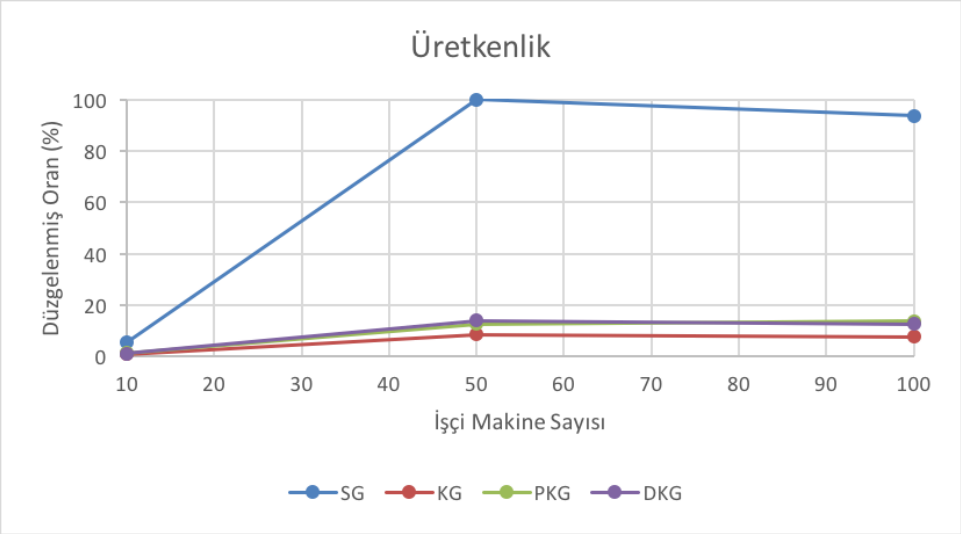
Şekil 39. Standart Sapma değerlerinin İşçi Birim sayılarına göre değişimi

Şekil 39'da görüldüğü üzere, İşçi Birim sayısının artması, Standart Sapma değerleri kapsamında, SG yönteminde bir değişikliğe yol açmamıştır. Diğer yöntemlerde ise düşüşe yol açmıştır. Özellikle İşçi Birim sayısının 10'dan 50'ye çıkartılmasıyla Standart Sapma değerlerinde ciddi düşüşler gözlemlenirken, 100'e çıkartılmasıyla 50'ye çıkartılmasına nazaran daha az bir düşüş gözlemlenmiştir. Bu kapsamda, İşçi Birim sayısının artmasıyla Standart Sapma değerlerinin daha da düşeceği ve böylece yükün makinelere daha dengeli dağıtılarak daha yüksek performans ve verimlilik elde edileceği öngörülmektedir.



Şekil 40. Dağıtım Maliyeti değerlerinin İşçi Birim sayılarına göre değişimi

Şekil 40'da görüldüğü üzere, İşçi Birim sayısının artması, Dağıtım Maliyeti değerleri kapsamında, KG, PKG ve DKG yöntemlerinde bir değişikliğe yol açmazken, SG yönteminde ise artışa sebebiyet vermiştir. KG, PKG ve DKG yöntemleri benzer değerler üretirken, SG yönteminde İşçi Birim sayısının artmasıyla Dağıtım Maliyeti değerinin de arttığı gözlemlenmiştir. Dağıtım Maliyeti değerinin artmasıyla ilişkili verilerin toplanmasının maliyeti artmaktadır ve dolayısıyla bu durum performansı da etkilemektedir. Bu kapsamda, KG, PKG ve DKG yöntemlerinin tercih edilmesi durumunda İşçi Birim sayısının arttırılmasına bağlı olarak Dağıtım Maliyetinin değişmesi beklenmezken, performans ve verimliliğinin azalmayacağı öngörülmektedir. Bununla birlikte, SG yönteminde ise, İşçi Birim sayısının artışıyla doğru orantılı olarak Dağıtım Maliyetinin artması beklendiğinden performans kayıpları ve verimliliğin azalacağı öngörülmektedir.



Şekil 41. Üretkenlik değerlerinin İşçi Birim sayılarına göre değişimi

Şekil 41'de görüldüğü üzere, İşçi Birim sayısının artması, Üretkenlik değerleri kapsamında, KG, PKG ve DKG yöntemlerinde bir değişikliğe yol açmazken, SG yönteminde ise artışa sebebiyet vermiştir. KG, PKG ve DKG yöntemleri benzer değerler üretirken, SG yönteminde İşçi Birim sayısının artmasıyla Üretkenlik değerinin de arttığı gözlemlenmiştir. Bu kapsamda, KG, PKG ve DKG yöntemlerinin tercih edilmesi durumunda İşçi Birim sayısının arttırılmasına bağlı olarak Üretkenlik değerinin değişmesi beklenmezken, performans ve verimliliğinin azalmayacağı öngörülmektedir. Bununla birlikte, durumlu verilerin hedeflenmediği SG yönteminde ise, İşçi Birim sayısının artışıyla doğru orantılı olarak Üretkenlik değerinin artması beklendiğinden performans artışı ve verimliliğin artacağı öngörülmektedir. Yalnız, İşçi Birim sayısının 50'den 100'e çıkartıldığı durumda Üretkenlik değerinde düşüş gözlemlenmiştir. Bu nedenle, SG yönteminde dahi belirli bir sayıdan daha fazla İşçi Birim olmasının sistem performansını olumsuz yönde etkileyebileceği de görülmüştür.

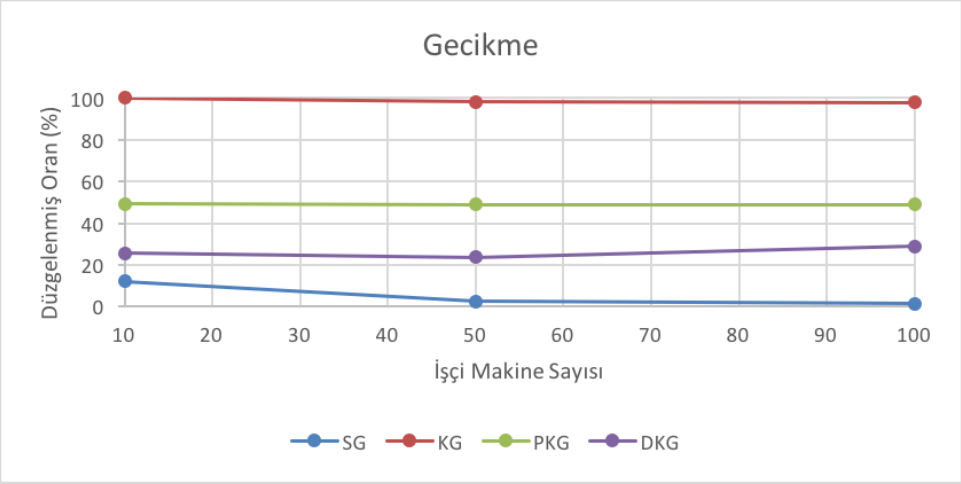
6.5.3.3. country-skew-r80 veri kümesi

country-skew-r80 veri kümesi (kayıt sayısı = 10,000,000, çarpıklık = %80) ile yapılan deneyin sonuçları Çizelge 12'de verilmiştir.

Çizelge 12. country-skew-r80 veri kümesi ile yapılan deneyin sonuçları

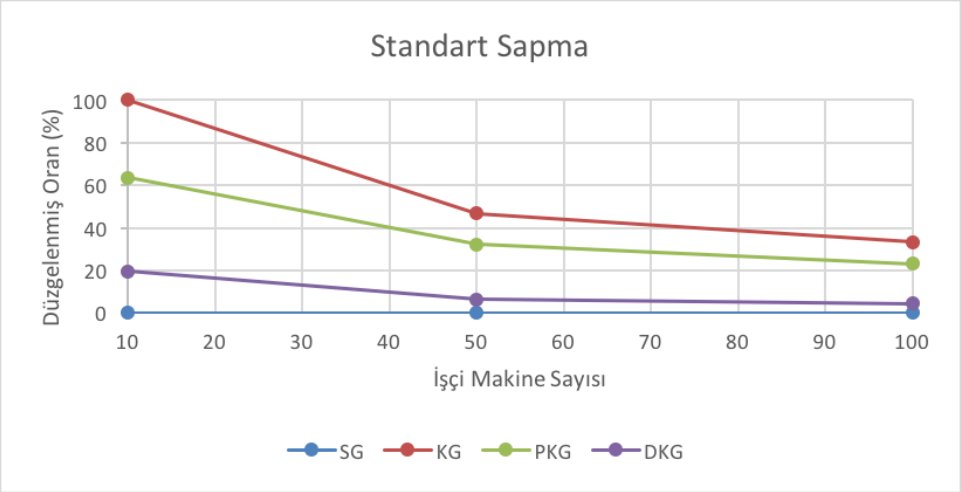
YÖNTEM	İŞÇİ MAKİNE SAYISI	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	10	1,044,141	0.0000	10.0000	9,579
KG	10	8,567,183	24.0180	1.0000	1,167
PKG	10	4,223,407	15.2432	1.6078	2,368
DKG	10	2,183,984	4.6728	2.8431	4,581
SG	50	208,826	0.0000	50.0000	48,077
KG	50	8,413,662	11.2140	1.0000	1,189
PKG	50	4,186,233	7.7659	1.8873	2,389
DKG	50	2,021,693	1.4947	3.1176	4,948
SG	100	118,596	0.0000	100.0000	84,746
KG	100	8,389,245	7.9699	1.0000	1,192
PKG	100	4,188,985	5.5780	1.9608	2,388
DKG	100	2,471,210	1.0904	2.3039	4,047

%80 çarpıklığa sahip country veri kümesinde, İşçi Birim sayısındaki artışın Gecikme, Dağıtım Maliyeti ve Üretkenlik değerlerinde değişikliğe sebebiyet vermediği, Standart Sapma değerinde ise azalış gösterdiği gözlemlenmiştir. Gözlemlere ait grafikler ve detaylı açıklamalar ise aşağıda verilmiştir.



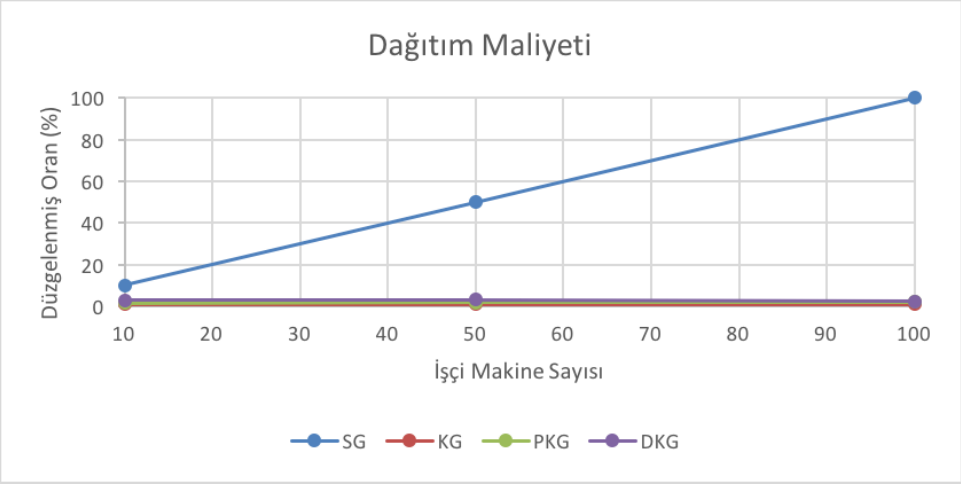
Şekil 42. Gecikme değerlerinin İşçi Birim sayılarına göre değişimi

Şekil 42'de görüldüğü üzere, Kaynak Birim sayısının artması, Gecikme değerleri kapsamında, KG ve PKG yöntemlerinde değişikliğe yol açmazken, DKG yönteminde 100 İşçi Birime çıkıldığı durumda çok hafif bir artışa yol açmıştır. Bununla beraber, SG yönteminde artan İşçi Birim sayısıyla birlikte Gecikme değerlerinin düştüğü gözlemlenmiştir. Bu kapsamda, çalışan makine sayısının arttırılmasının sistemin performansında ve verimliliğinde önemli bir iyileşme sağlamayacağı öngörülmektedir.



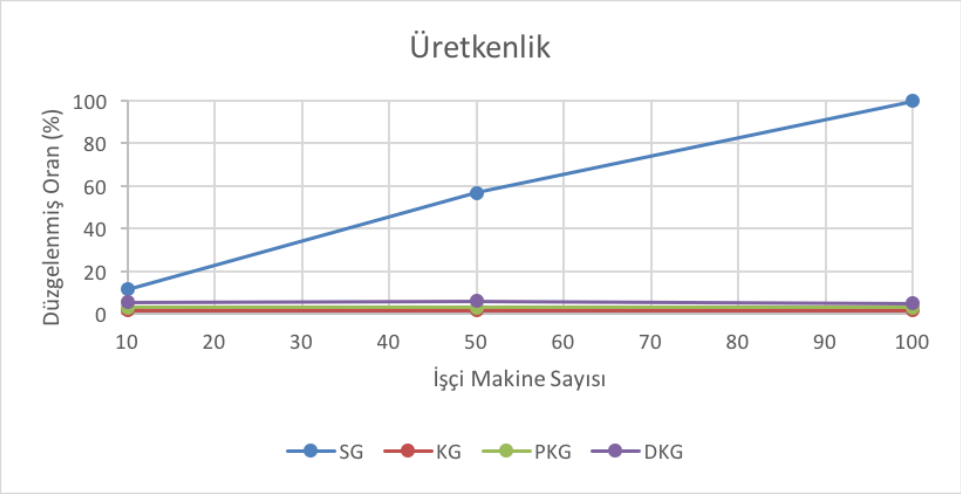
Şekil 43. Standart Sapma değerlerinin İşçi Birim sayılarına göre değişimi

Şekil 43'de görüldüğü üzere, İşçi Birim sayısının artması, Standart Sapma değerleri kapsamında, SG yönteminde bir değişikliğe yol açmamıştır. Diğer yöntemlerde ise düşüşe yol açmıştır. Özellikle İşçi Birim sayısının 10'dan 50'ye çıkartılmasıyla Standart Sapma değerlerinde ciddi düşüşler gözlemlenmişken, 100'e çıkartılmasıyla 50'ye çıkartılmasına kıyasla daha az bir düşüş gözlemlenmiştir. Bu kapsamda, İşçi Birim sayısının artmasıyla Standart Sapma değerlerinin daha da düşeceği ve böylece yükün makinelere daha dengeli dağıtılarak daha yüksek performans ve verimlilik elde edileceği öngörülmektedir.



Şekil 44. Dağıtım Maliyeti değerlerinin İşçi Birim sayılarına göre değişimi

Şekil 44'de görüldüğü üzere, İşçi Birim sayısının artması, Dağıtım Maliyeti değerleri kapsamında, KG, PKG ve DKG yöntemlerinde bir değişikliğe yol açmazken, SG yönteminde ise artışa sebebiyet vermiştir. KG, PKG ve DKG yöntemleri benzer değerler üretirken, SG yönteminde İşçi Birim sayısının artmasıyla Dağıtım Maliyeti değerinin de arttığı gözlemlenmiştir. Dağıtım Maliyetinin değerinin artmasıyla ilişkili verilerin toplanmasının maliyeti artmaktadır ve dolayısıyla bu durum performansı da etkilemektedir. Bu kapsamda, KG, PKG ve DKG yöntemlerinin tercih edilmesi durumunda İşçi Birim sayısının arttırılmasına bağlı olarak Dağıtım Maliyetinin değişmesi beklenmezken, performans ve verimliliğinin azalmayacağı öngörülmektedir. Bununla birlikte, SG yönteminde ise, İşçi Birim sayısının artışıyla doğru orantılı olarak Dağıtım Maliyetinin artması beklendiğinden performans kayıpları ve verimliliğin azalacağı öngörülmektedir.



Şekil 45. Üretkenlik değerlerinin İşçi Birim sayılarına göre değişimi

Şekil 45'te görüldüğü üzere, İşçi Birim sayısının artması, Üretkenlik değerleri kapsamında, KG, PKG ve DKG yöntemlerinde bir değişikliğe yol açmazken, SG yönteminde ise artışa sebebiyet vermiştir. KG, PKG ve DKG yöntemleri benzer değerler üretirken, SG yönteminde İşçi Birim sayısının artmasıyla Üretkenlik değerinin de arttığı gözlemlenmiştir. Bu kapsamda, KG, PKG ve DKG yöntemlerinin tercih edilmesi durumunda İşçi Birim sayısının arttırılmasına bağlı olarak Üretkenlik değerinin değişmesi beklenmezken, performans ve verimliliğinin azalmayacağı öngörülmektedir. Bununla birlikte, durumlu verilerin hedeflenmediği SG yönteminde ise, İşçi Birim sayısının artışıyla doğru orantılı olarak Üretkenlik değerinin artması beklendiğinden performans artışı ve verimliliğin artacağı öngörülmektedir.

6.5.4. Kaynak Birim Sayısının Değişiminin Etkisi

Deney, gerçek ve yapay veri kümeleri üzerinde 10 adet İşçi Birim karşılığında, 5, 10, 15 ve 20 Kaynak Birim ile çalıştırılmıştır. Bu kapsamda twitter-election, wikipedia-pageviews-by-lang ve country veri kümeleri kullanılmış olup, yöntemlerin artan Kaynak Birim sayıları karşısındaki performansları gözlemlenmiştir.

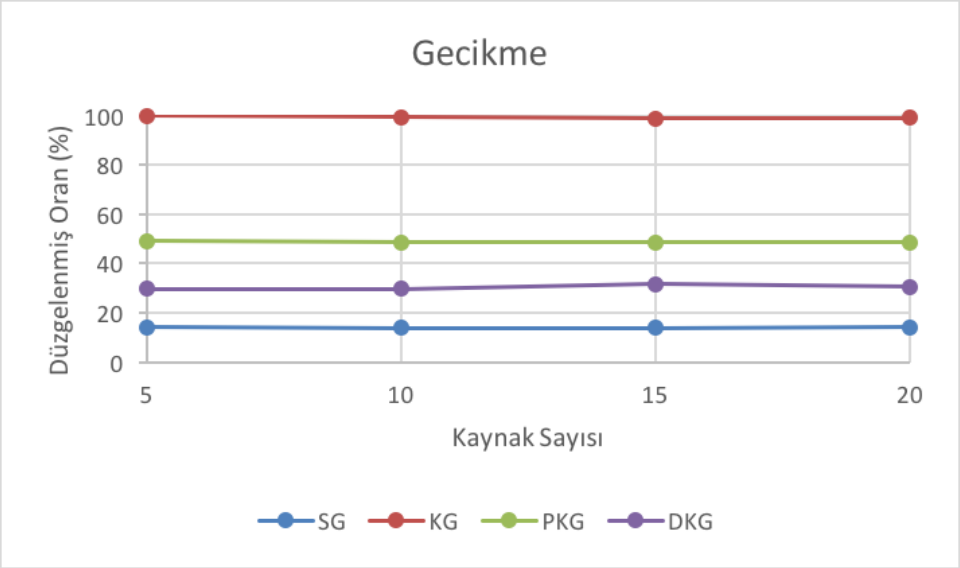
6.5.4.1. twitter-election veri kümesi

twitter-election veri kümesi (kayıt sayısı = 5,316,612, çarpıklık = %68) ile yapılan deneyin sonuçları Çizelge 13'te verilmiştir.

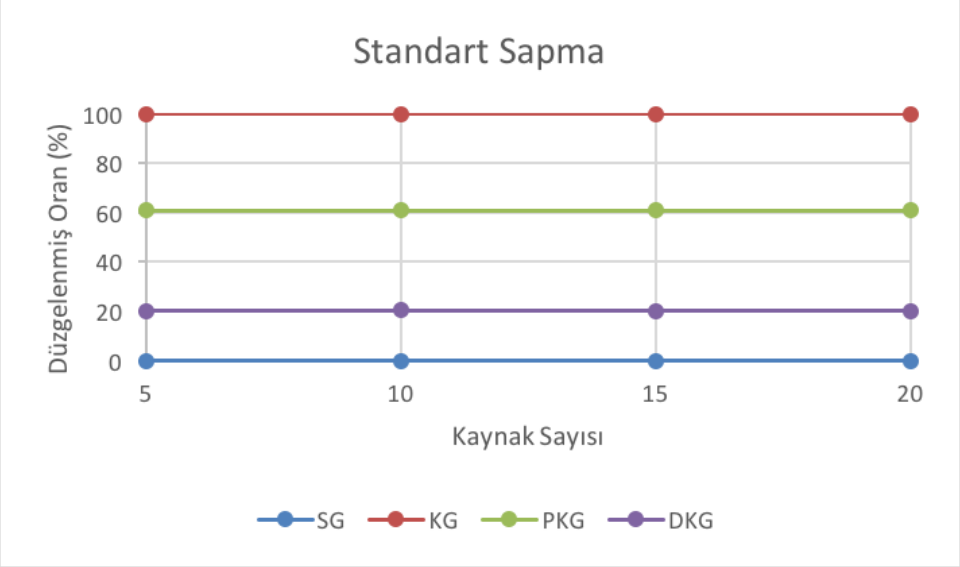
Çizelge 13. twitter-election veri kümesi ile yapılan deneyin sonuçları

YÖNTEM	KAYNAK BİRİM SAYISI	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	5	559,329	0.0001	2.4491	9,511
KG	5	3,955,045	20.3202	1.0000	1,344
PKG	5	1,941,879	12.3513	1.1647	2,739
DKG	5	1,174,979	4.0972	1.2414	4,529
SG	10	548,933	0.0000	2.4499	9,702
KG	10	3,925,956	20.3202	1.0000	1,355
PKG	10	1,924,469	12.3513	1.1662	2,763
DKG	10	1,179,683	4.1409	1.2416	4,509
SG	15	548,613	0.0001	2.4511	9,702
KG	15	3,911,011	20.3202	1.0000	1,359
PKG	15	1,918,636	12.3513	1.1658	2,772
DKG	15	1,254,250	4.0924	1.2364	4,240
SG	20	559,379	0.0000	2.4508	9,511
KG	20	3,920,839	20.3202	1.0000	1,356
PKG	20	1,925,265	12.3513	1.1677	2,762
DKG	20	1,210,381	4.1031	1.2394	4,394

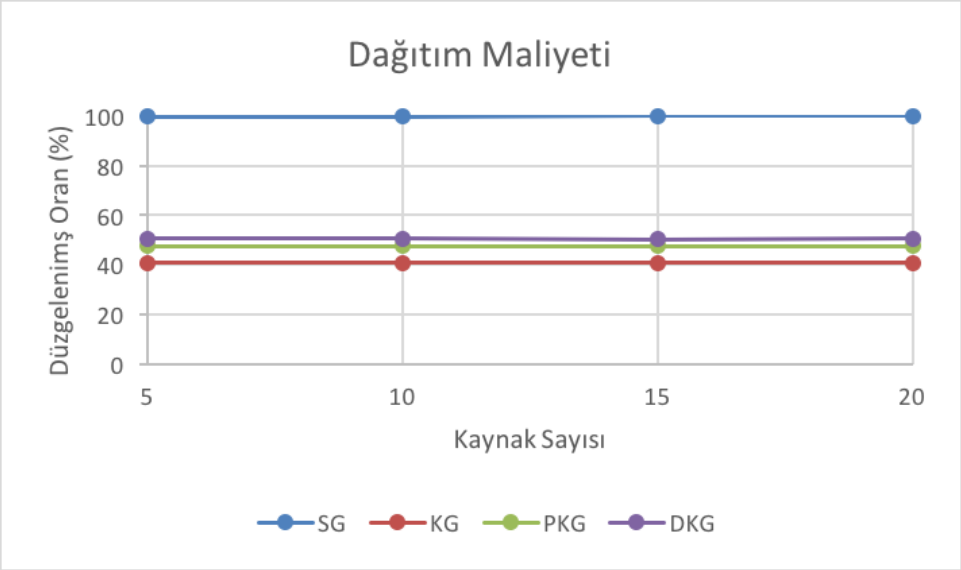
Şekil 46, Şekil 47, Şekil 48 ve Şekil 49'da görüleceği üzere, %68 çarpıklığa sahip twitter-election veri kümesinde, Kaynak Birim sayısındaki artışın Standart Sapma, Gecikme, Dağıtım Maliyeti ve Üretkenlik değerlerinde herhangi bir değişikliğe sebebiyet vermediği gözlemlenmiştir. Bu kapsamda, bu veri kümesi için Kaynak Birim sayısındaki artışın sistemin performansına ve verimliliğine olumlu veya olumsuz etki yapmayacağı öngörülmektedir. Gözlemlere ait grafikler aşağıda verilmiştir.



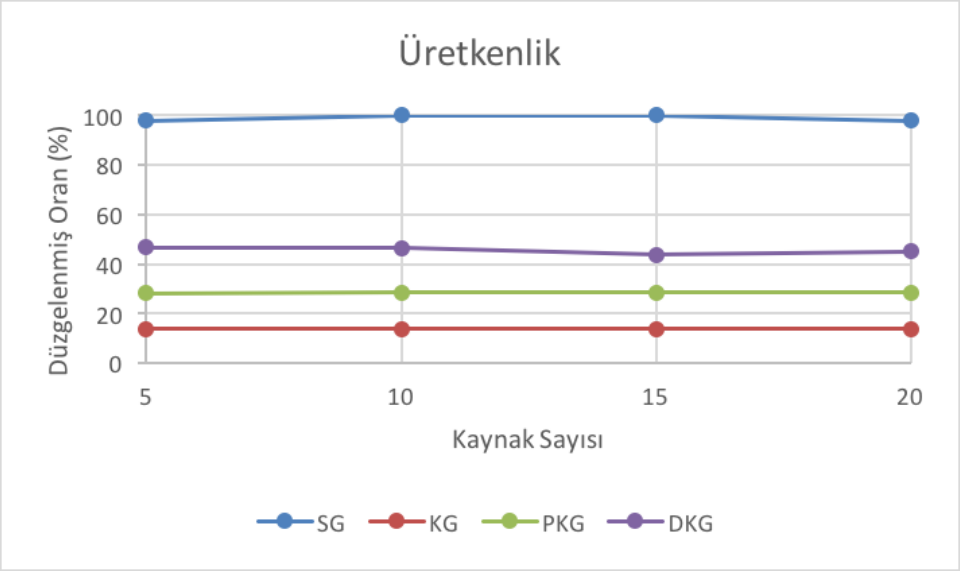
Şekil 46. Gecikme değerlerinin Kaynak Birim sayılarına göre değişimi



Şekil 47. Standart Sapma değerlerinin Kaynak Birim sayılarına göre değişimi



Şekil 48. Dağıtım Maliyeti değerlerinin Kaynak Birim sayılarına göre değişimi



Şekil 49. Üretkenlik değerlerinin Kaynak Birim sayılarına göre değişimi

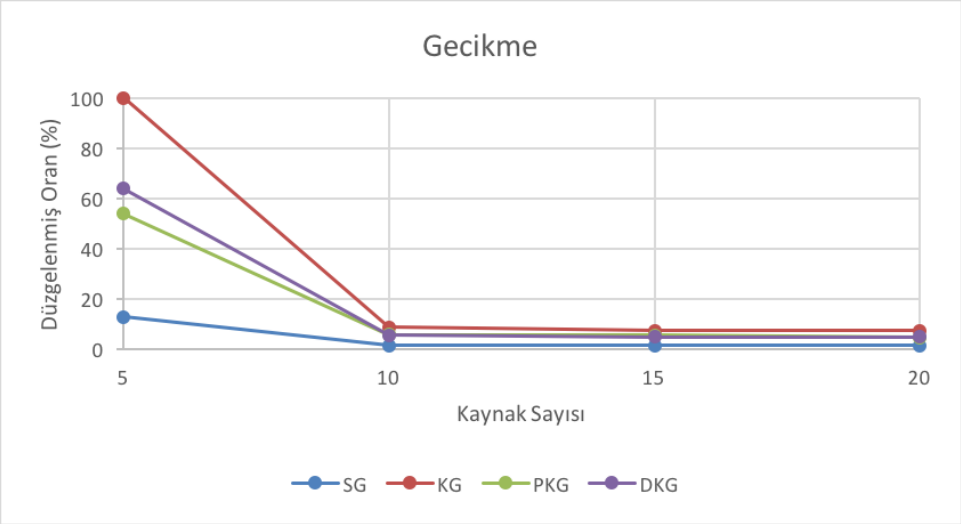
6.5.4.2. wikipedia-pageviews-by-lang veri kümesi

wikipedia-pageviews-by-lang veri kümesi (kayıt sayısı = 588,214,391, çarpıklık = %27) ile yapılan deneyin sonuçları Çizelge 14'te verilmiştir.

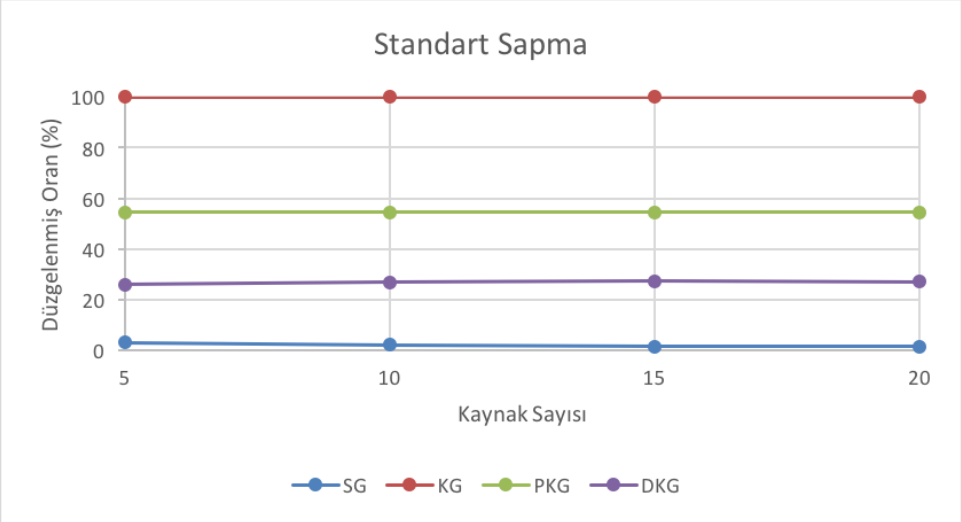
Çizelge 14. wikipedia-pageviews-by-lang veri kümesi ile yapılan deneyin sonuçları

YÖNTEM	KAYNAK BİRİM SAYISI	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	5	16,934,614	0.4049	9.9238	34,736
KG	5	130,874,178	13.9257	1.0000	4,495
PKG	5	70,501,163	7.5734	1.4317	8,343
DKG	5	83,789,184	3.5902	1.9460	7,020
SG	10	1,970,231	0.2919	9.9175	298,586
KG	10	11,514,162	13.9257	1.0000	51,087
PKG	10	7,713,736	7.5798	1.4413	76,263
DKG	10	7,536,199	3.7239	1.9302	78,054
SG	15	1,954,666	0.2023	9.9270	301,031
KG	15	9,893,652	13.9257	1.0000	59,458
PKG	15	7,400,215	7.5863	1.4667	79,488
DKG	15	6,430,996	3.7803	1.9333	91,480
SG	20	1,940,267	0.1916	9.9238	303,203
KG	20	9,707,930	13.9257	1.0000	60,597
PKG	20	6,241,169	7.5718	1.4667	94,250
DKG	20	6,592,289	3.7602	1.9333	89,232

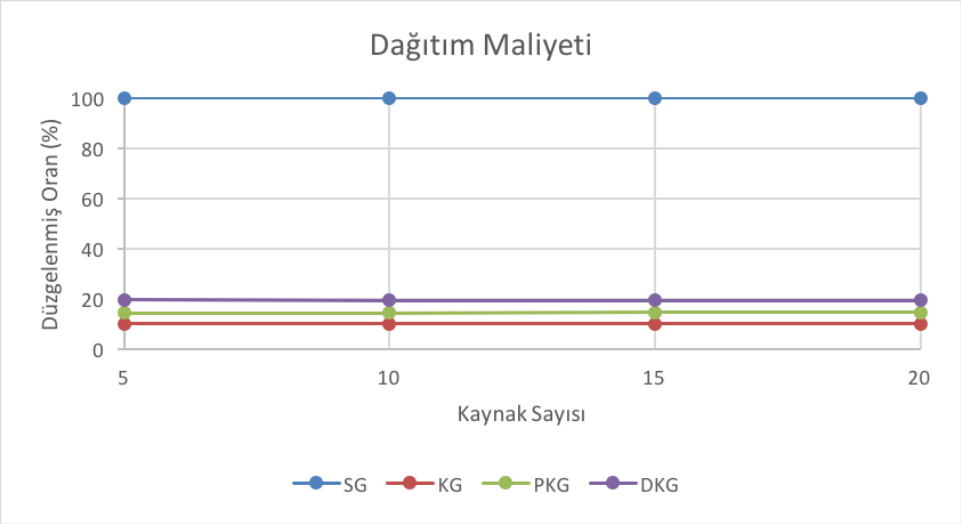
Şekil 50, Şekil 51, Şekil 52 ve Şekil 53'de görüleceği üzere, %27 çarpıklığa sahip wikipedia-pageviews-by-lang veri kümesinde, Kaynak Birim sayısındaki artışın Standart Sapma ve Dağıtım Maliyeti değerlerinde herhangi bir değişikliğe sebebiyet vermediği gözlemlenmiştir. Öte yandan, Kaynak Birim sayısının 5'ten 10'a çıkartılması durumunda Gecikme sürelerinde düşüş Üretkenlik değerinde ise artış gözlemlenmiştir ancak 10'dan sonraki artışlarda sabitlenmiş ve herhangi bir değişiklik görülmemiştir. Bu kapsamda, bu veri kümesi için Kaynak Birim sayısındaki 10'a kadar olan artışın sistemin performansına ve verimliliğine az da olsa olumlu etki yaratacağı ancak daha fazla Kaynak Birim kullanıldığı durumda sistemin performansına ve verimliliğine olumlu veya olumsuz etki yapmayacağı öngörülmektedir. Gözlemlere ait grafikler aşağıda verilmiştir.



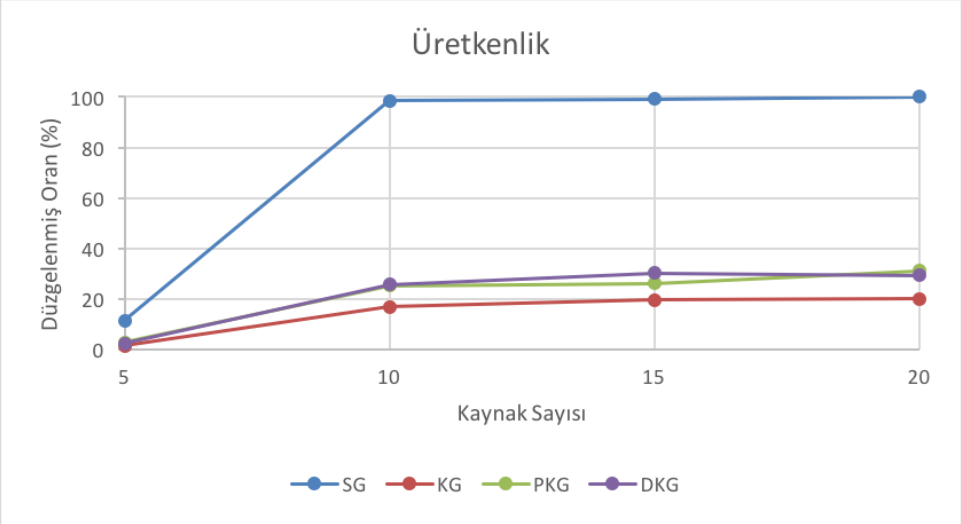
Şekil 50. Gecikme değerlerinin Kaynak Birim sayılarına göre değişimi



Şekil 51. Standart Sapma değerlerinin Kaynak Birim sayılarına göre değişimi



Şekil 52. Dağıtım Maliyeti değerlerinin Kaynak Birim sayılarına göre değişimi



Şekil 53. Üretkenlik değerlerinin Kaynak Birim sayılarına göre değişimi

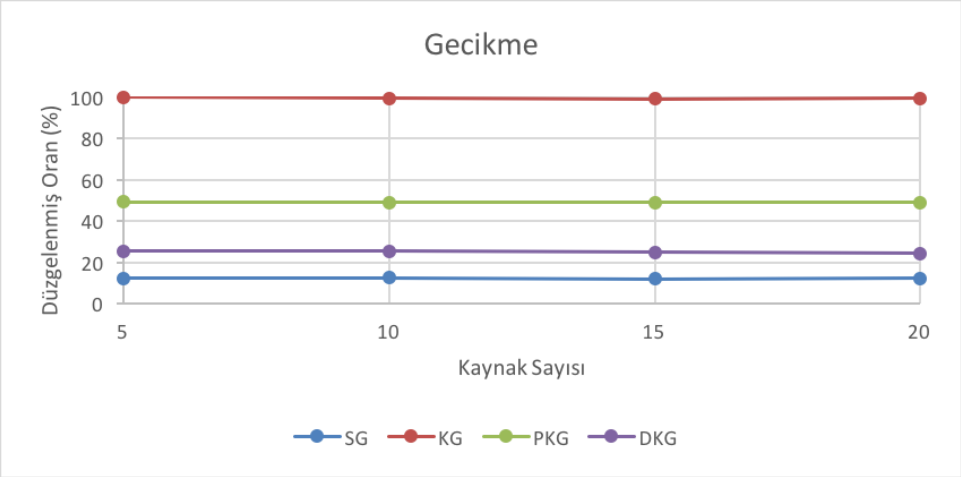
6.5.4.3. country-skew-r80 veri kümesi

country-skew-r80 veri kümesi (kayıt sayısı = 10,000,000, çarpıklık = %80) ile yapılan deneyin sonuçları Çizelge 15'te verilmiştir.

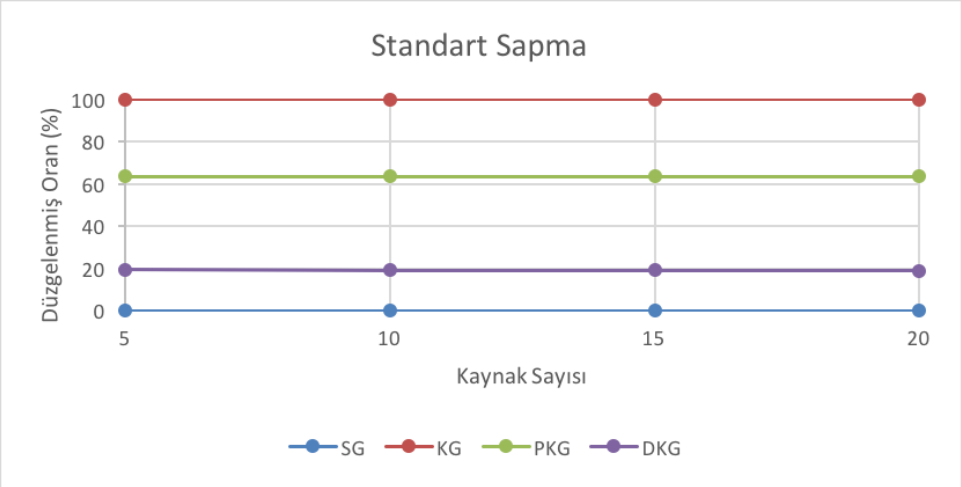
Çizelge 15. country-skew-r80 veri kümesi ile yapılan deneyin sonuçları

YÖNTEM	KAYNAK BİRİM SAYISI	GECİKME (milisaniye)	STANDART SAPMA	DAĞITIM MALİYETİ	ÜRETKENLİK (kayıt/saniye)
SG	5	1,044,141	0.0000	10.0000	9,579
KG	5	8,567,183	24.0180	1.0000	1,167
PKG	5	4,223,407	15.2432	1.6078	2,368
DKG	5	2,183,984	4.6728	2.8431	4,581
SG	10	1,074,954	0.0000	10.0000	9,311
KG	10	8,516,027	24.0180	1.0000	1,174
PKG	10	4,207,565	15.2432	1.6078	2,377
DKG	10	2,184,182	4.5951	2.7794	4,579
SG	15	1,036,551	0.0000	10.0000	9,653
KG	15	8,495,319	24.0180	1.0000	1,177
PKG	15	4,192,432	15.2432	1.6078	2,385
DKG	15	2,127,882	4.6142	2.7696	4,701
SG	20	1,042,998	0.0000	10.0000	9,597
KG	20	8,522,165	24.0180	1.0000	1,173
PKG	20	4,209,169	15.2432	1.6029	2,376
DKG	20	2,092,568	4.5139	2.8382	4,780

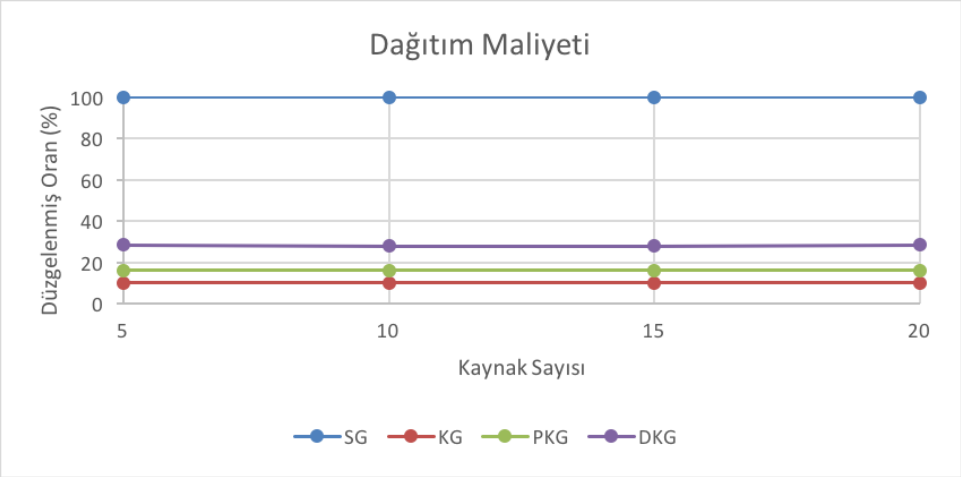
Şekil 54, Şekil 55, Şekil 56 ve Şekil 57'de görüleceği üzere, %80 çarpıklığa sahip country veri kümesinde, Kaynak Birim sayısındaki artışın Standart Sapma, Gecikme, Dağıtım Maliyeti ve Üretkenlik değerlerinde herhangi bir değişikliğe sebebiyet vermediği gözlemlenmiştir. Bu kapsamda, bu veri kümesi için Kaynak Birim sayısındaki artışın sistemin performansına ve verimliliğine olumlu veya olumsuz etki yapmayacağı öngörülmektedir. Gözlemlere ait grafikler aşağıda verilmiştir.



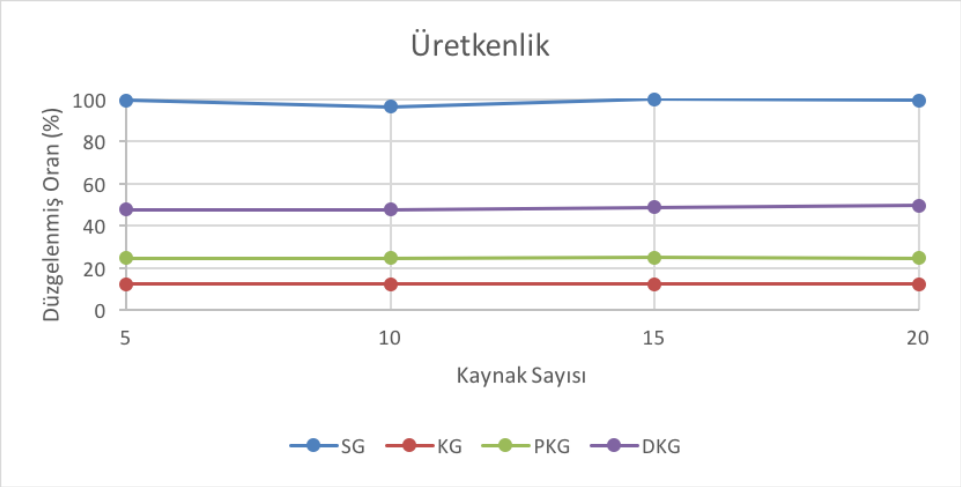
Şekil 54. Gecikme değerlerinin Kaynak Birim sayılarına göre değişimi



Şekil 55. Standart Sapma değerlerinin Kaynak Birim sayılarına göre değişimi



Şekil 56. Dağıtım Maliyeti değerlerinin Kaynak Birim sayılarına göre değişimi



Şekil 57. Üretkenlik değerlerinin Kaynak Birim sayılarına göre değişimi

6.6. Karşılaşılan Zorluklar

Çalışmanın gerçekleştirilmesi ve deneylerin yapılması sırasında karşılaşılan güçlüklere ve yaşanan problemlere bu kesimde yer verilecektir.

Öncelikle, üzerinde çalışabilecek büyük verilerin bulunmasında zorlanılmıştır. Günümüzde birçok veriye İnternet üzerinden kolayca ulaşılabilmektedir. Özellikle Büyük Veri alanında veri sağlayan Weka, OpenData, DBpedia ve Freebase gibi birçok kuruluş vardır. Yine de, belirli konularla ilgili özel veri kümelerinin bulunabilmes kolay olmamaktadır. Çalışma kapsamında çarpık veri içeren büyük hacimli veriler arandığından, sağlanan bu verilerin birçoğu yetersiz olmuştur. Bu nedenle metin tabanlı veriler içeren, yüksek hacme sahip Twitter ve Wikipedia verilerinin kullanılması tercih edilmiştir.

Veri kümelerini uygulamaya vermeden önce, ön işlemlerden geçirme ihtiyacımız oldu. Farklı veri kümeleri farklı karakter kodları kullandığından, uygulamanın okuyacağı şekilde dosyaların karakter kodlamalarını UTF-8'e ayarlamamız gerekti. Ayrıca, veriler tüm ülke dillerini kapsadığından, UTF-8 kapsamında olmayan birçok farklı karakterle karşılaştık. Bu karakterleri içeren verileri, uygulamanın çalışmasını etkilememesine rağmen, parazit veri olarak düşünerek ayıkladık.

Veri kümelerinde yer alan her bir satır düzenli ve standart değildi. Verilerin tek bir şablonu ve formatı olmadığından, bütün satırlar için aynı varsayımda bulunamadık. Örneğin satırların büyük çoğunluğu boşlukla ayrılmış 4 kelimeden oluşurken, bazı satırların 1 ve 2 kelimeden oluştuğuna şahit olduk. Yine bu tür satırları, parazit veri olarak tanımladık ve ayıkladık.

Veri çeşitliği çok fazla olduğunda, yani verinin içerisinde birbirinden farklı çok fazla anahtar kelime geçtiği durumlarda, sistemin üretkenliğinde düşüş gözlemledik. İlingenin yapısı gereği, işlenen veriler bir süre biriktirilip, periyodik olarak gruplar halinde bir sonraki İşçi Birime gönderilmektedirler. Bu biriktirilme süresince toplanan verilerin çeşitliliğinin arttığı durumlarda, İşçi Birimler arası aktarılan verinin boyutu da oldukça büyümektedir. Bu da hem ağ trafiğini hem de ilingenin performansını olumsuz etkileyebilmektedir.

DKG yöntemi, yapısı gereği sık geçen anahtar kelimeleri bellekte sakladığından, veri çeşitliliğinin artması durumunda kullanılan bellek miktarında artış olabilmektedir.

İlinge çalışırken ara sonuçların Kafka'ya yazılması sağlanabilmektedir. Ancak anlık olarak işlenen veri hacminin artması sonucu, Kafka'ya atılacak birim zamandaki veri miktarı da oldukça yüksek değerlere ulaşabilmektedir. Bu da işletim sistemi seviyesinde yüksek I/O oranlarına çıkılması ve CS (context-switch) değerlerinin artması ile sonuçlanmaktadır. Dolayısıyla uygulamanın çalışması da dolaylı olarak etkilenebilmekte ve üretkenliği azalarak performansında ciddi düşüşler meydana gelebilmektedir.

Ayrıca uzun süre yapılan testler sırasında, yüksek hacimli verilerin diske sürekli yazılıp silinmesi neticesinde, disklerde birtakım sorunlar çıkmıştır ve disklerin bir kısmı bozulmuştur. Bu sebeple yüksek veri trafiği alarak diskin yoğun kullanacağı sistemlerde mutlaka verilerin yedekli tutulması gerekmektedir.

Sonuç olarak, bütün bu problemlerden de deneyimler kazandık ve Büyük Verinin, aynı zamanda, büyük problemler ve büyük sorumluluklar da demek olduğunu öğrendik.

7. SONUÇ

Bu çalışma kapsamında, durumlu çarpık veri kümeleri üzerinde çalışan dağıtık mimarideki akan-veri işleme yöntemleri incelenmiştir. Akabinde, dağıtık mimarilerde yükün makinelere dengeli dağıtılabilmesi için yapılan çalışmalar araştırılmış ve öğrenilmiştir. SG (Karışık Gruplama) yöntemi toplam yükü kümedeki bütün makinelere en iyi şekilde dağıttığı halde, durumlu veriler söz konusu olduğunda Dağıtım Maliyeti yüksek olduğundan tercih edilememektedir. Durumlu veriler için aynı anahtar değeriyle gelen verileri aynı makinelere yönlendiren KG (Anahtar Gruplama) yöntemi tercih edilebilmektedir. KG yöntemi de, çarpık verilerin geldiği, yani belirli tür anahtarların daha çok geldiği durumlarda yükü belirli makinelerde yoğunlaştırdığından, sistemin verimsiz çalışmasına ve yüksek gecikme sürelerine ulaşmasına sebebiyet vermektedir. KG yönteminin verimsiz kaldığı noktada, özellikle çarpık verilerin geldiği durumlar için PKG (Parçalı Anahtar Gruplama) yöntemi önerilmiştir. Bu yöntemle, KG yöntemi ile aynı anahtar değerine sahip olan çok-öğeliler sadece bir makineye gönderilirken, PKG yöntemi ile seçilen iki makineye gönderilmektedir. Her çok-öğeli için iki makine seçilip, aralarında daha az yüke sahip olan makineye veriler gönderilmektedir. Sistemin tamamına hâkim olan bir yapı olmadığından, her makine kendi yerel yük dağılımını tutmakta ve kendisinin yükü dağıttığı makinelerdeki oranları bilebilmektedir. Yerel yük bilgisi ile çalışan bir çözüm olmasına rağmen, sistemin tamamına baktığımızda oldukça başarılı ve tatmin edici sonuçlar elde edilmiştir ve üretkenlikte %60'a, gecikme süresinde ise %45'e varan iyileşmenin gözlemlendiği belirtilmiştir. Ancak PKG yöntemi de verinin yüksek derecede çarpıklığa sahip olduğu durumlarda iyi performans gösterememektedir. Ayrıca bu yöntemde, yükün her daim iki makineye dağıtılacağının garantisi verilememektedir. Çünkü dağıtım yapılacak olan hedef makineler, dağıtılacak olan verinin iki farklı karma fonksiyonuna girmesi sonucu mevcut makine sayısıyla mod'u alınarak tespit edilmektedir. Aynı hedef makine değerlerinin üretildiği durumlarda yük tek makinede yoğunlaşacaktır. Öte yandan, PKG yöntemi, yapısı itibariyle yükü en fazla iki adet makineye paylaştırabilmektedir. Bu da verinin çok az bir kısmının çok fazla yoğunluğa sahip olması durumunda, kümedeki makinelerden çok azının aktif olarak kullanılmasına sebebiyet vermektedir. Böyle bir senaryoda, makine sayısının

arttırılması da sistemin performansını arttırmayacaktır. Bu çalışma kapsamında ise, veri çarpıklığının yüksek olduğu bu tür senaryolar için, dağıtım yapılacak makine sayısını iki ile kısıtlamayarak, yoğun olan verileri tespit eden ve bu verileri yoğunluklarıyla orantılı olarak daha çok makineye dağıtan DKG (Dinamik Anahtar Gruplama) adlı yöntem önerilmektedir.

Yöntemler, verinin içeriğine bağlı olarak değişkenlik göstermektedir. Durumsuz veri kümelerinde SG yöntemi tercih edilmeliyken, durumlu veri kümeleri için, eğer verinin içeriği bilinmiyorsa KG yöntemi, ölçeklenebilir ve tahmin edilebilir olmadığından tercih edilmemelidir. Bu kapsamda, KG üzerine geliştirilmiş olan ve KG yöntemine kıyasla daha performanslı çalışan PKG veya DKG yöntemleri tercih edilmelidir. Bu kapsamda, mevcuttaki SG, KG ve PKG yöntemi ile önerilen DKG yönteminin performansları, durumlu veri kümeleri ile dağıtık mimarideki akan-veri işleme motorları üzerinde test edilerek karşılaştırılmıştır. Yapılan deneylerin sonucunda, %30'luk çarpıklığın bulunduğu durumda PKG yöntemine kıyasla DKG yönteminin Gecikmede %7 ve Üretkenlikte %8 oranında daha olumlu sonuçlar verdiği gözlemlenmiştir. Bununla birlikte, DKG yöntemiyle %80 ve üzeri yüksek çarpıklık oranlarında Gecikmede %48 ve Üretkenlikte %93'e varan iyileşmeler gözlemlenmiştir. Bu kapsamda, durumlu veri kümesinin içeriği önceden bilinmiyorsa veya tahmin edilemiyorsa, PKG yöntemi tercih edilebilirken, veri çarpıklığının %30 ve üzeri olmasının beklendiği koşullarda, DKG yöntemi tercih edilmelidir.

8. KAYNAKÇA

[1] Doug Laney, 3D Data Management: Controlling Data Volume, Velocity and Variety, **2001**, http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

[2] Moore Law, Gordon Moore, https://en.wikipedia.org/wiki/Moore%27s_law (Haziran, **2017**).

[3] Samza, http://samza.apache.org/ (Haziran, **2017**).

[4] S4, https://github.com/s4 (Haziran, **2017**).

[5] Spark, http://spark.apache.org/ (Haziran, **2017**).

[6] Storm, http://storm.apache.org/ (Haziran, **2017**).

[7] Heron, https://twitter.github.io/heron/ (Haziran, **2017**).

[8] Kafka, https://kafka.apache.org/ (Haziran, **2017**).

[9] Kafka streams, https://kafka.apache.org/documentation/streams (Haziran, **2017**).

[10] Hazelcast Jet, http://jet.hazelcast.org/ (Haziran, **2017**).

[11] Kinesis, https://aws.amazon.com/kinesis/ (Haziran, **2017**).

[12] Zookeeper, https://zookeeper.apache.org/ (Haziran, **2017**).

[13] Justin Ellingwood, Big Data Frameworks Compared, https://www.digitalocean.com/community/tutorials/hadoop-storm-samza-spark-and-flink-big-data-frameworks-compared (**2017**).

[14] Jim Scott, Stream Processing Everywhere, https://mapr.com/blog/stream-processing-everywhere-what-use/ (**2017**).

[15] Michael G. Noll, Understanding the Parallelism of a Storm Topology, http://www.michael-noll.com/blog/2012/10/16/understanding-the-parallelism-of-a-storm-topology/ (June, **2017**)

[16] M. A. Shah, J. M. Hellerstein, S. Chandrasekaran, and M. J. Franklin, “Flux: An adaptive partitioning operator for continuous query systems,” in ICDE, **2003**, pp. 25–36.

[17] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. B. Zdonik, “Scalable distributed stream processing,” in CIDR, vol. 3, **2003**, pp. 257–268.

[18] Y. Xing, S. Zdonik, and J.-H. Hwang, “Dynamic load distribution in the borealis stream processor,” in ICDE, **2005**, pp. 791–802.

[19] B. Gedik, “Partitioning functions for stateful data parallelism in stream processing,” The VLDB Journal, pp. 1–23, **2013**.

[20] C. Balkesen, N. Tatbul, and M. T. Özsu, “Adaptive input admission and management for parallel stream processing,” in DEBS. ACM, **2013**, pp. 15–26.

[21] R. Castro Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch, “Integrating scale out and fault tolerance in stream processing using operator state management,” in SIGMOD, **2013**, pp. 725–736.

[22] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, “Balanced allocations,” SIAM J. Comput., vol. 29, no. 1, pp. 180–200, **1999**.

[23] M. Mitzenmacher, “The power of two choices in randomized load balancing,” IEEE Trans. Parallel Distrib. Syst., vol. 12, no. 10, pp. 1094–1104, **2001**.

[24] J. Byers, J. Considine, and M. Mitzenmacher, “Geometric generalizations of the power of two choices,” in SPAA, **2003**, pp. 54–63.

[25] M. Mitzenmacher, R. Sitaraman et al., “The power of two random choices: A survey of techniques and results,” in Handbook of Randomized Computing, **2001**, pp. 255–312.

[26] M. A. U. Nasir, G. De Francisci Morales, D. García-Soriano, N. Kourtellis and M. Serafini, "The power of both choices: Practical load balancing for distributed stream processing engines," 2015 IEEE 31st International Conference on Data Engineering, Seoul, **2015**, pp. 137-148.

[27] Bifet, Albert; Holmes, Geoff; Kirkby, Richard; Pfahringer, Bernhard (**2010**). "MOA: Massive online analysis". The Journal of Machine Learning Research 99: 1601–1604.

[28] Morales, Gianmarco De Francisci, and Albert Bifet. "SAMOA: scalable advanced massive online analysis." Journal of Machine Learning Research 16.1 (**2015**): 149-153.

[29] B. Lohrmann, P. Janacik and O. Kao, "Elastic Stream Processing with Latency Guarantees," 2015 IEEE 35th International Conference on Distributed Computing Systems, Columbus, OH, **2015**, pp. 399-410.

[30] Z. Abbas, “Streaming Graph Partitioning : Degree Project in Distributed Computing at KTH Information and Communication Technology,” Dissertation, **2016**.

[31] Lei Cao , Elke A. Rundensteiner, High performance stream query processing with correlation-aware partitioning, Proceedings of the VLDB Endowment, v.7 n.4, p.265-276, December **2013**.

[32] Cagri Balkesen, Nesime Tatbul, M. Tamer Özsu, Adaptive input admission and management for parallel stream processing, Proceedings of the 7th ACM international conference on Distributed event-based systems, June 29-July 03, **2013**, Arlington, Texas, USA

[33] Sanjeev Kulkarni , Nikunj Bhagat , Maosong Fu , Vikas Kedigehalli , Christopher Kellogg , Sailesh Mittal , Jignesh M. Patel , Karthik Ramasamy , Siddarth Taneja, Twitter Heron: Stream Processing at Scale, Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, May 31-June 04, **2015**, Melbourne, Victoria, Australia

[34] Nicoló Rivetti, Emmanuelle Anceaume, Yann Busnel, Leonardo Querzoni, Bruno Sericola. Online Scheduling for Shuffle Grouping in Distributed Stream Processing Systems Research Paper. ACM/IFIP/USENIX Middleware 2016 , Dec **2016**, Trento, Italy.

[35] Raul Castro Fernandez , Matteo Migliavacca , Evangelia Kalyvianaki , Peter Pietzuch, Integrating scale out and fault tolerance in stream processing using operator state management, Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, June 22-27, **2013**, New York, New York, USA

[36] Shadi A. Noghabi , Kartik Paramasivam , Yi Pan , Navina Ramesh , Jon Bringhurst , Indranil Gupta , Roy H. Campbell, Samza: stateful scalable stream processing at LinkedIn, Proceedings of the VLDB Endowment, v.10 n.12, August **2017**

[37] T. Heinze, V. Pappalardo, Z. Jerzak and C. Fetzer, "Auto-scaling techniques for elastic data stream processing," 2014 IEEE 30th International Conference on Data Engineering Workshops, Chicago, IL, **2014**, pp. 296-302.

[38] V. Gulisano, R. Jiménez-Peris, M. Patiño-Martínez, C. Soriente and P. Valduriez, "StreamCloud: An Elastic and Scalable Data Streaming System," in IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 12, pp. 2351-2365, Dec. **2012**.

[39] B. Gedik, S. Schneider, M. Hirzel and K. L. Wu, "Elastic Scaling for Data Stream Processing," in IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 6, pp. 1447-1463, June **2014**.

[40] Zacheilas N., Zygouras N., Panagiotou N., Kalogeraki V., Gunopulos D. (**2016**) Dynamic Load Balancing Techniques for Distributed Complex Event Processing Systems. In: Jelasity M., Kalyvianaki E. (eds) Distributed Applications and Interoperable Systems. Lecture Notes in Computer Science, vol 9687. Springer, Cham

[41] Scott Schneider, Joel Wolf, Kirsten Hildrum, Rohit Khandekar, and Kun-Lung Wu. **2016**. Dynamic Load Balancing for Ordered Data-Parallel Regions in Distributed Streaming Systems. In Proceedings of the 17th International Middleware Conference (Middleware '16). ACM, New York, NY, USA, Article 21, 14 pages.

[42] V. Gil-Costa, N. Hidalgo, E. Rosas and M. Marin, "A Dynamic Load Balance Algorithm for the S4 Parallel Stream Processing Engine," 2016 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW), Los Angeles, CA, **2016**, pp. 19-24.

[43] R. Shah, B. Veeravalli and M. Misra, "On the Design of Adaptive and Decentralized Load Balancing Algorithms with Load Estimation for Computational Grid Environments," in IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 12, pp. 1675-1686, Dec. **2007**.

[44] M. Hanif and C. Lee, "An efficient key partitioning scheme for heterogeneous MapReduce clusters," 2016 18th International Conference on Advanced Communication Technology (ICACT), Pyeongchang, **2016**, pp. 364-367.

[45] Gothai Ekambaram and Balasubramanie Palanisamy, "A Modified Key Partitioning for BigData Using MapReduce in Hadoop," Journal of Computer Science 2015, 11 (3): 490.497

[46] Ibrahim, S., Jin, H., Lu, L. et al. Peer-to-Peer Netw. Appl. (**2013**) 6: 409.

[47] Nicoló Rivetti , Leonardo Querzoni , Emmanuelle Anceaume , Yann Busnel , Bruno Sericola, Efficient key grouping for near-optimal load balancing in stream processing systems, Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, June 29-July 03, **2015**, Oslo, Norway

[48] R. Wang and K. Chiu, "A stream partitioning approach to processing large scale distributed graph datasets," 2013 IEEE International Conference on Big Data, Silicon Valley, CA, **2013**, pp. 537-542.

[49] N. Xu, B. Cui, L. Chen, Z. Huang and Y. Shao, "Heterogeneous Environment Aware Streaming Graph Partitioning," in IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 6, pp. 1560-1572, June 1 **2015**.

[50] Pedro Bizarro , Shivnath Babu , David DeWitt , Jennifer Widom, Content-based routing: different plans for different data, Proceedings of the 31st international conference on Very large data bases, August 30-September 02, **2005**, Trondheim, Norway

[51] Guoli Li , Vinod Muthusamy , Hans-Arno Jacobsen, Adaptive content-based routing in general overlay topologies, Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware, December 01-05, **2008**, Leuven, Belgium

[52] S. Bhowmik, M. A. Tariq, L. Hegazy and K. Rothermel, "Hybrid Content-Based Routing Using Network and Application Layer Filtering," 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), Nara, **2016**, pp. 221-231.

[53] Muhammad Adnan Tariq , Boris Koldehofe , Kurt Rothermel, Efficient content-based routing with network topology inference, Proceedings of the 7th ACM international conference on Distributed event-based systems, June 29-July 03, **2013**, Arlington, Texas, USA

[54] Sukanya Bhowmik , Muhammad Adnan Tariq , Jonas Grunert , Kurt Rothermel, Bandwidth-efficient content-based routing on software-defined networks, Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, June 20-24, **2016**, Irvine, California

[55] Muhammad Shafique, Adaptive Content-based Routing using Subscription Subgrouping in Structured Overlays, **2016**. (https://arxiv.org/abs/1604.06853)

[56] YongChul Kwon , Magdalena Balazinska , Bill Howe , Jerome Rolia, SkewTune: mitigating skew in mapreduce applications, Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, May 20-24, **2012**, Scottsdale, Arizona, USA

[57] X. Zhang, H. Chen and F. Hu, "Back Propagation Grouping: Load Balancing at Global Scale When Sources Are Skewed," 2017 IEEE International Conference on Services Computing (SCC), Honolulu, HI, **2017**, pp. 426-433.

[58] Y. Le, J. Liu, F. Ergün and D. Wang, "Online load balancing for MapReduce with skewed data input," IEEE INFOCOM 2014 - IEEE Conference on Computer Communications, Toronto, ON, **2014**, pp. 2004-2012.

[59] L. Cheng, S. Kotoulas, T. E. Ward and G. Theodoropoulos, "Efficiently Handling Skew in Outer Joins on Distributed Systems," 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, IL, **2014**, pp. 295-304.

[60] Junhua Fang, Rong Zhang, Tom Z.J. Fu, Zhenjie Zhang, Aoying Zhou, and Junhua Zhu. **2017**. Parallel Stream Processing Against Workload Skewness and Variance. In Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '17). ACM, New York, NY, USA, 15-26.

[61] Muhammad Anis Uddin Nasir, Hiroshi Horii, Marco Serafini, Nicolas Kourtellis, Rudy Raymond, Sarunas Girdzijauskas, Takayuki Osogami, Load Balancing for Skewed Streams on Heterogeneous Cluster, https://arxiv.org/abs/1705.09073

[62] M. A. U. Nasir, G. D. F. Morales, N. Kourtellis and M. Serafini, "When two choices are not enough: Balancing at scale in Distributed Stream Processing," 2016

IEEE 32nd International Conference on Data Engineering (ICDE), Helsinki, **2016**, pp. 589-600.

[63] K. Talwar and U. Wieder, “Balanced allocations: the weighted case,” in STOC, 2007, pp. 256–265.

[64] M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen, “Parallel Randomized Load Balancing,” in STOC, 1995, pp. 119–130.

[65] C. Lenzen and R. Wattenhofer, “Tight bounds for parallel randomized load balancing: Extended abstract,” in STOC, 2011, pp. 11–20.

[66] G. Park, “A Generalization of Multiple Choice Balls-into-bins,” in PODC, 2011, pp. 297–298.

[67] JVM Heap Modeli, http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html (Haziran, **2017**).

[68] Centos, https://www.centos.org/ (Haziran, **2017**).

[69] Confluent, https://www.confluent.io/ (Haziran, **2017**).

[70] InfluxDB, https://www.influxdata.com/ (Haziran, **2017**).

[71] Grafana, https://grafana.com/ (Haziran, **2017**).

[72] Source Code: DKG - Dynamic Key Grouping (Haziran, **2017**), https://github.com/odalabasmaz/DynamicKeyGrouping

[73] Dataset: Tweets During the 2016 Election, http://anuragprasad.com/blog/twitter_election/, (Haziran, **2017**).

[74] Dataset: Wikipedia Clickstream, https://figshare.com/articles/Wikipedia_Clickstream/1305770 (Haziran, **2017**).

[75] Dataset: Wikipedia Pageviews by Language, https://dumps.wikimedia.org/other/pageviews/ (Haziran, **2017**).

9. EKLER

9.1. Veri Kümesi Örnekleri

Her veri kümesinden örnek girdi, çıktı.

- twitter-election


```
{
  "created_at":"Mon Nov 07 18:27:13 +0000 2016",
  "id":795694108429996000,
  "id_str":"795694108429996034",
  "text":"RT @DebraMessing: Please WATCH &amp;amp; SHARE #VOTE https://t.co/2zWV5na0FT",
  "entities":{"hashtags":[{"text":"VOTE","indices":[43,48]}]},
  "lang":"en",
```


- twitter-ticker

```
1383264000      AAPL
```

- wikipedia-clickstream

```
Oceanography  Aristotle        link      19
```

- wikipedia-pageviews

```
1199195421      http://en.wikipedia.org/wiki/Research
```

- wikipedia-pageviews-by-lang

```
tr Lale_Devri 2 0
```

- country

```
Turkey
```

9.2. Kaynak Kodlar

Bu kesimde mevcutta bulunan yöntemler ile önerilen DKG yöntemine ait kaynak kodları yer almaktadır.

```
public class ShuffleGrouping implements CustomStreamGrouping, Serializable {
   private static final long serialVersionUID = -8251986623724271906L;
   private int index;
   private List<Integer> targetTasks;

   @Override
   public void prepare(WorkerTopologyContext context, GlobalStreamId stream, List<Integer>
targetTasks) {
      this.index = -1;
      this.targetTasks = targetTasks;
   }

   @Override
   public List<Integer> chooseTasks(int taskId, List<Object> values) {
      List<Integer> boltIds = new ArrayList<>(1);
      setNextIndex();
      boltIds.add(targetTasks.get(index));
      return boltIds;
   }

   private void setNextIndex() {
      index = ++index % targetTasks.size();
   }
}
```

Kod 1. SG Yöntemi

```
public class KeyGrouping implements CustomStreamGrouping, Serializable {
   private static final long serialVersionUID = 4347587663034269976L;
   private List<Integer> targetTasks;
   private HashFunction h1 = Hashing.murmur3_128(13);

   @Override
   public void prepare(WorkerTopologyContext context, GlobalStreamId stream, List<Integer>
targetTasks) {
      this.targetTasks = targetTasks;
   }

   @Override
   public List<Integer> chooseTasks(int taskId, List<Object> values) {
      List<Integer> boltIds = new ArrayList<>(1);
      if (!values.isEmpty()) {
         String key = values.get(0).toString(); // assume key is the first field
         int selected = (int) (Math.abs(h1.hashBytes(key.getBytes()).asLong()) %
this.targetTasks.size());
         boltIds.add(targetTasks.get(selected));
      }
      return boltIds;
   }
}
```

Kod 2. KG Yöntemi

```
public class PartialKeyGrouping implements CustomStreamGrouping, Serializable {
   private static final long serialVersionUID = -4473798373140003 53L;
   private List<Integer> targetTasks;
   private long[] targetTaskStats;
   private HashFunction h1 = Hashing.murmur3_128(13);
   private HashFunction h2 = Hashing.murmur3_128(17);

   @Override
   public void prepare(WorkerTopologyContext context, GlobalStreamId stream, List<Integer>
targetTasks) {
      this.targetTasks = targetTasks;
      targetTaskStats = new long[this.targetTasks.size()];
   }

   @Override
   public List<Integer> chooseTasks(int taskId, List<Object> values) {
      List<Integer> boltIds = new ArrayList<>(1);
      if (values.size() > 0) {
         String str = values.get(0).toString(); // assume key is the first field
         int firstChoice = (int) (Math.abs(h1.hashBytes(str.getBytes()).asLong()) %
this.targetTasks.size());
         int secondChoice = (int) (Math.abs(h2.hashBytes(str.getBytes()).asLong()) %
this.targetTasks.size());
         int selected = targetTaskStats[firstChoice] > targetTaskStats[secondChoice] ?
secondChoice : firstChoice;
         boltIds.add(targetTasks.get(selected));
         targetTaskStats[selected]++;
      }
      return boltIds;
   }
}
```

Kod 3. PKG Yöntemi

```
public class DynamicKeyGrouping implements CustomStreamGrouping, ObjectObserver, Serializable {
   private static final Logger LOGGER = LoggerFactory.getLogger(DynamicKeyGrouping.class);
   private static final long serialVersionUID = 3981182647363702 94L;

   private long startingTime;
   private long warmUpDuration = 15_000L;     // ms (default: 30 sec)
   private int numberOfInitialTasks = 2;
   private int numberOfAvailableTasks;
   private long checkInterval = 60 * 1000L;

   private long numOfItems;
   private long[] targetTaskStats;
   private List<Integer> targetTasks;

   private int loadToScaleUp;
   private int loadToScaleDown;

   private final KeySpace keySpace;
   private final Map<String, Integer> workerCountSizeMap;

   public DynamicKeyGrouping() {
      String id = this.toString();
      LOGGER.info("DKG-D: {}", id);
      keySpace = new KeySpace();
      workerCountSizeMap = new HashMap<>();
   }

   public DynamicKeyGrouping(int distinctKeyCounts) {
      String id = this.toString();
      LOGGER.info("DKG: {}", id);
      keySpace = new KeySpace(distinctKeyCounts);
      workerCountSizeMap = new HashMap<>();
   }
…
```

```
…
   @Override
   public void prepare(WorkerTopologyContext context, GlobalStreamId streamId, List<Integer>
targetTasks) {
      this.targetTasks = targetTasks;
      this.targetTaskStats = new long[targetTasks.size()];
      this.numberOfAvailableTasks = targetTasks.size();
      initThresholds();
      initStartingTime();
      initKeySpaceManagement();
   }

   private void initThresholds() {
      double idealLoad = (double) 100 / numberOfAvailableTasks;
      loadToScaleUp = (int) (idealLoad + Math.sqrt(idealLoad));
      loadToScaleDown = (int) (idealLoad - Math.sqrt(idealLoad));
      LOGGER.info("LoadToScaleUp: {} and LoadToScaleDown: {}", loadToScaleUp, loadToScaleDown);
   }

   private void initStartingTime() {
      if (startingTime == 0L) {
         startingTime = System.currentTimeMillis();
      }
   }

   private void initKeySpaceManagement() {
      LOGGER.info("KeySpaceManagement is being initialized...");
      ExecutorService executor = Executors.newSingleThreadExecutor();
      executor.submit(new KeySpaceManager(keySpace));
//    executor.submit(new KeySpaceGC(keySpace));
   }

   private boolean isWarmUp() {
      long now = System.currentTimeMillis();
      long timePassed = now - startingTime;
      return timePassed > warmUpDuration;
   }

   @Override
   public List<Integer> chooseTasks(int taskId, List<Object> values) {
      List<Integer> chosenTasks = new ArrayList<>(1);
      if (!values.isEmpty()) {
         String key = values.get(0).toString();
         synchronized (keySpace) {
            Integer chosen = chooseBestTask(key);       // index of best task
            Integer targetTask = targetTasks.get(chosen);
            chosenTasks.add(targetTask);
            handleKey(key, chosen, targetTask);
         }
      }
      return chosenTasks;
   }

   private void handleKey(String key, Integer chosen, Integer targetTask) {
      targetTaskStats[chosen]++;
      numOfItems++;
      keySpace.handleKey(key, targetTask);
   }
…
```

```
…
   private Integer chooseBestTask(String key) {
      long hashOfKey = DKGUtils.calculateHash(key);
      int targetWorkerIndex = normalizeIndex(hashOfKey);
      Integer workerCount = workerCountSizeMap.get(key);
      if (workerCount == null) {
         workerCount = numberOfInitialTasks;      //i.e. 2 tasks available in startup
      }

      // workers which has less load
      int numberOfLessLoad = 0;

      // select least loaded task as the best
      Integer bestTaskIndex = null;
      int minLoad = Integer.MAX_VALUE;
      for (int i = 0; i < workerCount; ++i) {
         int taskIndex = normalizeIndex((long) targetWorkerIndex + i);
         int load = getCurrentLoadOfTask(taskIndex);
         if (load < loadToScaleUp) {
            numberOfLessLoad++;
         }
         if (load < minLoad) {
            minLoad = load;
            bestTaskIndex = taskIndex;
         }
      }

      // wait for a while before scaling
      if (canCheckForScaling(key)) {
         LOGGER.info("Checking for scaling...");
         if (shouldScaleUp(key, minLoad)) {
            // check if it should scale up
            int newTaskIndex = normalizeIndex((long) targetWorkerIndex + workerCount);
            int newTaskLoad = getCurrentLoadOfTask(newTaskIndex);
            if (newTaskLoad < minLoad) {
               bestTaskIndex = newTaskIndex;
               workerCountSizeMap.put(key, workerCount + 1);
//             LOGGER.info("#WS: workerCountSizeMap.size() = {}", workerCountSizeMap.size());
               LOGGER.info("Scaling up for key: {}, to: {} workers at dkg: {} with
newTaskLoad/minLoad: {}/{}",
                     key, workerCount + 1, getObjectId(), newTaskLoad, minLoad);
            }
         } else if (shouldScaleDown(workerCount, numberOfLessLoad)) {
            // check if it should scale down
            workerCountSizeMap.put(key, workerCount - 1);
            LOGGER.info("Scaling down for key: {}, to: {} workers at dkg: {} for load: {}",
                  key, workerCount - 1, getObjectId(), minLoad);
            return chooseBestTask(key);
         }
      }

      return bestTaskIndex;
   }

   private boolean canCheckForScaling(String key) {
      KeyItem keyItem = keySpace.findKeyItem(key);
      if (keyItem == null) {
         return false;
      }

      long now = DKGUtils.getCurrentTimestamp();
      long latestCheckTime = keyItem.getLastCheckTimeForScaling();
      if (now - latestCheckTime > checkInterval) {
         keyItem.setLastCheckTimeForScaling(now);
         return true;
      }
      return false;
   }
…
```

```
…
   private boolean shouldScaleUp(String key, int minLoad) {
      // is warm up
      if (!isWarmUp()) {
         return false;
      }

      // check threshold
      if (minLoad < loadToScaleUp) {
         return false;
      }

      // is in old-gen
      if (!keySpace.inOldSpace(key)) {
         return false;
      }

      return true;
   }

   private boolean shouldScaleDown(int workerCount, int numberOfLessLoad) {
      return workerCount > 2 && numberOfLessLoad >= 2;
   }

   /**
    * calculates the local load of the task
    */
   private int getCurrentLoadOfTask(int task) {
      return numOfItems == 0 ? 0 : (int) ((100 * targetTaskStats[task]) / numOfItems);
   }

   private int normalizeIndex(long index) {
      return (int) (index % numberOfAvailableTasks);
   }
}
```

Kod 4. DKG: DynamicKeyGrouping

```
package com.orhundalabasmaz.storm.loadbalancer.grouping.dkg;

import org.apache.commons.lang3.StringUtils;

import java.util.LinkedHashSet;
import java.util.Set;

/**
 * @author Orhun Dalabasmaz
 */
public class KeyItem implements Comparable<KeyItem> {
   private String key;
   private long lastSeen;
   private long lastCheckTimeForScaling;
   private long count;
   private Set<Integer> targetTasks;

   public KeyItem(String key, int targetTask) {
      this.key = key;
      this.count = 1;
      this.lastSeen = System.currentTimeMillis();
      this.targetTasks = new LinkedHashSet<>();
      this.targetTasks.add(targetTask);
   }

…
```

```
…
   public String getKey() {
      return key;
   }

   public long getLastSeen() {
      return lastSeen;
   }

   public long getCount() {
      return count;
   }

   public int getTargetTasksCount() {
      return targetTasks.size();
   }

   public long getLastCheckTimeForScaling() {
      return lastCheckTimeForScaling;
   }

   public void setLastCheckTimeForScaling(long time) {
      this.lastCheckTimeForScaling = time;
   }

   public void appearedAgain(int targetTask) {
      ++count;
      targetTasks.add(targetTask);
      lastSeen = System.currentTimeMillis();
   }

   @Override
   public int compareTo(KeyItem item) {
      return (int) (count - item.count);
   }

   @Override
   public int hashCode() {
      if (StringUtils.isBlank(key)) return 0;
      return key.hashCode();
   }

   @Override
   public boolean equals(Object obj) {
      if (this == obj) return true;
      if (obj == null || !(obj instanceof KeyItem)) return false;
      KeyItem item = (KeyItem) obj;
      return this.key.equals(item.key);
   }
}
```

Kod 5. DKG: KeyItem

```
package com.orhundalabasmaz.storm.loadbalancer.grouping.dkg;

import com.orhundalabasmaz.storm.utils.DKGUtils;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.Serializable;
import java.util.*;

public class KeySpace implements Serializable {
   private static final Logger LOGGER = LoggerFactory.getLogger(KeySpace.class);
   private Map<String, KeyItem> babySpace;          // max heap
   private Map<String, KeyItem> teenageSpace;       // max heap
   private Map<String, KeyItem> oldSpace;           // max heap
…
```

```
…
   private final int OLD_MAX_SIZE;
   private final int TEENAGE_MAX_SIZE;
   private final int BABY_MAX_SIZE;

   private static final Comparator<KeyItem> DESC_COMPARATOR = (k1, k2) -> (int) (k2.getCount() -
k1.getCount());

   public KeySpace() {
      this(100);
   }

   public KeySpace(int distinctKeyCounts) {
      OLD_MAX_SIZE = (int) (distinctKeyCounts * (10 / (double) 100));
      TEENAGE_MAX_SIZE = (int) (distinctKeyCounts * (40 / (double) 100));
      BABY_MAX_SIZE = (int) (distinctKeyCounts * (50 / (double) 100));

      oldSpace = new HashMap<>(OLD_MAX_SIZE);
      teenageSpace = new HashMap<>(TEENAGE_MAX_SIZE);
      babySpace = new HashMap<>(BABY_MAX_SIZE);
   }

   public void handleKey(String key, int targetTask) {
      KeyItem item = findKeyItem(key);
      if (item != null) {
         item.appearedAgain(targetTask);
      } else {
         item = new KeyItem(key, targetTask);
         babySpace.put(key, item);
      }
   }

   public KeyItem findKeyItem(String key) {
      KeyItem item = oldSpace.get(key);
      if (item != null) return item;
      item = teenageSpace.get(key);
      if (item != null) return item;
      item = babySpace.get(key);
      return item;
   }

   private KeyItem findKeyItemInList(String key, Set<KeyItem> keyItemList) {
      if (StringUtils.isBlank(key)) {
         return null;
      }
      for (KeyItem item : keyItemList) {
         if (item == null) {
            LOGGER.error("item cannot be null!");
            continue;
         }
         if (key.equals(item.getKey())) {
            return item;
         }
      }
      return null;
   }

   public boolean inBabySpace(String key) {
      return babySpace.containsKey(key);
   }

   public boolean inTeenageSpace(String key) {
      return teenageSpace.containsKey(key);
   }

   public boolean inOldSpace(String key) {
      return oldSpace.containsKey(key);
   }

…
```

```
…
   public void promoteToTeenageSpace() {
      LOGGER.debug("promoteToTeenageSpace");
      List<KeyItem> babyList = new ArrayList<>(babySpace.values());
      List<KeyItem> teenageList = new ArrayList<>(teenageSpace.values());
      babyList.sort(DESC_COMPARATOR);
      babyList = DKGUtils.truncateList(babyList, BABY_MAX_SIZE);
      teenageList.sort(DESC_COMPARATOR);
      promoteToNextSpace(babyList, teenageList, TEENAGE_MAX_SIZE);

      babySpace.clear();
      DKGUtils.putListIntoMap(babyList, babySpace);
      teenageSpace.clear();
      DKGUtils.putListIntoMap(teenageList, teenageSpace);
   }

   public void promoteToOldSpace() {
      LOGGER.debug("promoteToOldSpace");
      List<KeyItem> teenageList = new ArrayList<>(teenageSpace.values());
      List<KeyItem> oldList = new ArrayList<>(oldSpace.values());
      teenageList.sort(DESC_COMPARATOR);
      oldList.sort(DESC_COMPARATOR);
      promoteToNextSpace(teenageList, oldList, OLD_MAX_SIZE);
      printKeyItems(oldList);

      teenageSpace.clear();
      DKGUtils.putListIntoMap(teenageList, teenageSpace);
      oldSpace.clear();
      DKGUtils.putListIntoMap(oldList, oldSpace);
   }

   private void printKeyItems(List<KeyItem> keyItems) {
      keyItems.sort(DESC_COMPARATOR);
      StringBuilder builder = new StringBuilder();
      keyItems.forEach(item -> builder
            .append(item.getKey()).append(",")
            .append(item.getCount()).append(",")
            .append(item.getTargetTasksCount()).append("\n"));
      LOGGER.info("#HOTTEST KEYS:\n{}", builder);
   }

   private void promoteToNextSpace(List<KeyItem> fromSpace, List<KeyItem> toSpace, int
toSpaceMaxSize) {
      int loop = 0;
      while (toSpace.size() < toSpaceMaxSize) {
         if (fromSpace.isEmpty()) return;
         KeyItem firstBaby = fromSpace.get(0);
         fromSpace.remove(0);
         toSpace.add(0, firstBaby);
         ++loop;
      }

      for (; loop < toSpaceMaxSize; ++loop) {
         if (fromSpace.isEmpty()) return;
         KeyItem firstBaby = fromSpace.get(0);
         int lastTeenIndex = toSpace.size() - 1;
         KeyItem lastTeen = toSpace.get(lastTeenIndex);
         if (firstBaby.getCount() > lastTeen.getCount()) {
            fromSpace.remove(0);
            toSpace.remove(lastTeenIndex);
            toSpace.add(0, firstBaby);
            fromSpace.add(lastTeen);
         } else {
            break;
         }
      }
   }
}
```

Kod 6. DKG: KeySpace

```java
package com.orhundalabasmaz.storm.loadbalancer.grouping.dkg;

import com.orhundalabasmaz.storm.utils.DKGUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * @author Orhun Dalabasmaz
 */
public class KeySpaceManager implements Runnable {
   private static final Logger LOGGER = LoggerFactory.getLogger(KeySpaceManager.class);
   private static final long TIME_INTERVAL = 15;
   private static final long CYCLE_CO UNT = 4;
   private boolean run = true;
   private final KeySpace keySpace;

   public KeySpaceManager(KeySpace keySpace) {
      this.keySpace = keySpace;
   }

   @Override
   public void run() {
      int count = 0;
      while (run) {
         ++count;

         synchronized (keySpace) {
            // rearrange keys in space
            keySpace.promoteToTeenageSpace();

            if (count == CYCLE COUNT) {
               count = 0;
               keySpace.promoteToOldSpace();
            }
         }

         // wait for next execution
         DKGUtils.sleepInSeconds(TIME_INTERVAL);
      }
   }

   public void terminate() {
      run = false;
      LOGGER.info("KeySpaceManager is terminated.");
   }
}
```

Kod 7. KeySpaceManager

10. ÖZGEÇMİŞ

Kimlik Bilgileri

Adı Soyadı : Orhun DALABASMAZ

Doğum Yeri : Ankara

Medeni Hali : Evli

E-posta : odalabasmaz@gmail.com

Adresi : Mehtap Sitesi A/36 Yaşamkent Mah. Çankaya / ANKARA

Eğitim

Lisans : Hacettepe Üniversitesi – Bilgisayar Mühendisliği

Yabancı Dil ve Düzeyi

İngilizce – İyi

İş Deneyimi

Cybersoft – 2010/06 – 2012/12

Innova – 2013/06 – 2017/06

OpsGenie – 2017/07 – (devam)

Deneyim Alanları

Java, AWS, DevOps, SRE

Tezden Üretilmiş Projeler ve Bütçesi

-

Tezden Üretilmiş Yayınlar

-

Tezden Üretilmiş Tebliğ ve/veya Poster Sunumu ile Katıldığı Toplantılar

-

HACETTEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
YÜKSEK LİSANS/DOKTORA TEZ ÇALIŞMASI ORJİNALLİK RAPORU

HACETTEPE ÜNİVERSİTESİ
FEN BİLİMLER ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI BAŞKANLIĞI'NA

Tarih: 13/02/2018

Tez Başlığı / Konusu: DİNAMİK ANAHTAR GRUPLAMA: DAĞITIK AKAN-VERİ KATARI İŞLEME SİSTEMLERİ İÇİN BİR YÜK DENGELEME ALGORİTMASI

Yukarıda başlığı/konusu gösterilen tez çalışmamın a) Kapak sayfası, b) Giriş, c) Ana bölümler d) Sonuç kısımlarından oluşan toplam 116 sayfalık kısmına ilişkin, 13/02/2018 tarihinde şahsım/tez danışmanım tarafından *Turnitin* adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı %1'dir.

Uygulanan filtrelemeler:

1- Kaynakça hariç
2- Alıntılar hariç/dâhil
3- 5 kelimeden daha az örtüşme içeren metin kısımları hariç

Hacettepe Üniversitesi Fen Bilimleri Enstitüsü Tez Çalışması Orjinallik Raporu Alınması ve Kullanılması Uygulama Esasları'nı inceledim ve bu Uygulama Esasları'nda belirtilen azami benzerlik oranlarına göre tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Gereğini saygılarımla arz ederim.

13/02/2018
Tarih ve İmza

Adı Soyadı:	ORHUN DALABASMAZ
Öğrenci No:	N10228431
Anabilim Dalı:	BİLGİSAYAR MÜHENDİSLİĞİ
Programı:	BİLGİSAYAR MÜHENDİSLİĞİ – YÜKSEK LİSANS
Statüsü:	☒ Y.Lisans ☐ Doktora ☐ Bütünleşik Dr.

DANIŞMAN ONAYI

UYGUNDUR.

DOÇ. DR. AHMET BURAK CAN