T.C.
HARRAN ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

APACHE SPARK VE GPU'NUN BÜYÜK VERİ ANALİZİNDE KULLANILMASI

Mehmet TURAN

ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANA BİLİM DALI

ŞANLIURFA
2019

Dr. Öğr. Üyesi Mehmet Emin TENEKECİ danışmanlığında, Mehmet TURAN'ın hazırladığı "**Apache Spark ve GPU'nun Büyük Veri Analizinde Kullanılması**" konulu bu çalışma 27/08/2019 tarihinde aşağıdaki jüri tarafından oy birliği / oy çokluğu ile Harran Üniversitesi Fen Bilimler Enstitüsü Elektrik Elektronik Mühendisliği Anabilim Dalı'nda YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

İmza

Danışman : Dr. Öğr. Üyesi Mehmet Emin TENEKECİ ……

Üye : Dr. Öğr. Üyesi Abdülkadir GÜMÜŞÇÜ ……

Üye : Dr. Öğr. Üyesi Faruk KÜRKER ……

Bu Tezin Elektrik Elektronik Mühendisliği Anabilim Dalında Yapıldığını ve Enstitümüz Kurallarına Göre Düzenlendiğini Onaylarım.

Doç. Dr. İsmail HİLALİ
Enstitü Müdürü

Not: Bu tezde kullanılan özgün ve başka kaynaktan yapılan bildirimlerin, çizelge, şekil ve fotoğrafların kaynak gösterilmeden kullanımı 5846 sayılı Fikir ve Sanat Eserleri Kanunundaki hükümlere tabidir.

İÇİNDEKİLER

Sayfa No

İÇİNDEKİLER ii
ÖZET i
ABSTRACT ii
TEŞEKKÜR iii
ŞEKİLLER DİZİNİ iv
ÇİZELGELER DİZİNİ v
SİMGELER ve KISALTMALAR DİZİNİ vi
1. GİRİŞ 1
1.1. Büyük Veri 5
1.1.1. Hacim 6
1.1.2. Çeşitlilik 6
1.1.3. Hız 7
1.1.4. Güvenirlilik 7
1.1.5. Değişkenlik 7
1.1.6. Geçerlilik 7
1.1.7. Görselleştirme 8
1.1.8. Değer 8
1.2. GPU 8
1.2.1. CUDA mimarisi 9
1.3. Apache Spark 11
1.3.1. Spark SQL ve DataFrame 13
1.4. cuDF ve BlazingSQL 15
1.5. Problem Tanımı 17
1.6. Çalışmanın Amacı 18
1.7. Çalışmanın Önemi 19
2. ÖNCEKİ ÇALIŞAMALAR 21
3. MATERYAL ve YÖNTEM 30
3.1. Materyal 30
3.1.1. Donanım 30
3.1.2. Yazılım 32
3.1.3. Veri 32
3.2. Yöntem 33
3.2.1. Analiz edilecek verilerin hazırlanması 33
3.2.2. Sorgu türlerinin belirlenmesi 34
3.2.3. Apache Spark sorgularının gerçekleştirilmesi 36
3.2.4. GPU sorgularının gerçekleştirilmesi 37
4. ARAŞTIRMA BULGULARI ve TARTIŞMA 40
4.1. Araştırma Bulguları 40
4.2. Tartışma 45
5. SONUÇLAR ve ÖNERİLER 47
5.1. Sonuçlar 47
5.2. Öneriler 48
KAYNAKLAR 50
ÖZGEÇMİŞ 56

ÖZET

Yüksek Lisans Tezi

APACHE SPARK VE GPU'NUN BÜYÜK VERİ ANALİZİNDE KULLANILMASI

Mehmet TURAN

Harran Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik Elektronik Mühendisliği Anabilim Dalı

DANIŞMAN: Dr. Öğr. Üyesi Mehmet Emin TENEKECİ
YIL:2019, Sayfa: 56

Günlük hayatın her alanında gerçekleşen dijital dönüşüm farklı problemlerin ve ihtiyaçların ortaya çıkmasına neden olmuştur. Bu dönüşüme paralel olarak artan veri çeşitliliği, verilerdeki tutarsızlık, verinin büyümesiyle verinin daha hızlı analiz edilebilmesi ihtiyaçtan ziyade bir zorunluluk haline gelmiştir. Geleneksel donanımlar ve Hadoop, Spark ve Hive gibi yazılım temelli çözümler ile büyük verilerin işlenmesi, analizi ve yönetimi sağlanmaktadır. Gelişmiş donanımların sistem kapasitesinin artırılmasına önemli katkıları vardır. Aynı şekilde, güçlü bir donanıma sahip sistemin veri tabanının tasarlanmasına ve performansına da çok büyük etkisi vardırır. GPU'ların artan veri boyutunun sorgulanmasında kullanılması büyük veriler için iyi bir alternatif olabilir. CPU yerine, çok fazla çekirdeğin ve yüksek bellek boyutu ile yüksek derecede paralelleştirme teknolojisinin kullanıldığı GPU'nun büyük veri sorgularının hızlandırılmasına olan etkileri araştırılmıştır. Bu kapsamda bellek-temelli büyük veri hesaplama yapısı olan Apache Spark ile GPU DataFrame kütüphanesi olan cuDF yapılarından yararlanılmıştır. Bu çalışmada, veri analizinde yaygın olarak kullanılan sıralama, gruplandırma ve filtreleme gibi sorguların gerçekleşme sürelerine bağlı olarak performans karşılaştırması yapılmıştır. Aynı sorgular CPU ve GPU üzerinde ayrı ayrı gerçekleştirilmiştir. Bu sorgular sonucunda, basit sorgularda Apache Spark ve GPU gerçekleşme süreleri bakımından benzer sonuçlar vermesine karşın yoğun hesaplama gerektiren birçok sorguda GPU 2x-6x arasında daha hızlı sonuç verirken, koşula dayalı filtreleme işleminde Apache Spark yaklaşık olarak 5x daha hızlı gerçekleştirmiştir.

ANAHTAR KELİMELER: GPU, apache spark, cuDF, büyük veri

ABSTRACT

MSc Thesis

USING APACHE SPARK AND GPU ON BIG DATA ANALYSIS

Mehmet TURAN

Harran University
Graduate School of Natural and Applied Sciences
Department of Electric Electronic Engineering

Supervisor: Assist. Prof. Dr. Mehmet Emin TENEKECİ
YEAR: 2019, Page: 56

Digital transformation in daily life has led to different problems and needs. Depending of this transformation, with increase in data diversity, data inconsistency and data growth, faster analysis of data has become necessity rather than need. Traditional hardware and software-based solutions such as Hadoop, Spark and Hive supply the processing, analysis and management of big data. Advanced hardwares affects increasing in system capacity significantly. Likewise, a well-equipped system has a huge impact on the design and performance of the database. Using GPUs to query the increasing data size can be a good alternative for big data. Instead of CPU, the effects of GPU, which use high degree parallelization technology with many cores and high memory size for acceleration, was investigated on big data queries. In this context, Apache Spark, which is a memory-based large data computation framework, and cuDF structures, which is a GPU DataFrame library, were utilized. In this study, the performance comparison of the queries which are commonly used in data analysis such as sorting, grouping and filtering is processed. The effects of GPU on big data queries with the cuDF library were examined. Same queries were used in separately both on CPU and GPU. As a result of these queries, Apache Spark and GPU yields similar results in simple queries, but in many queries which requires intensive computations, GPU provides faster results between 2x-6x. However, Apache Spark performs approximately 5x faster in conditional filtering.

KEY WORDS: GPU, apache spark, cuDF, big data

TEŞEKKÜR

Tez konusunun belirlenmesinde, yürütülmesinde ve çalışmamda önemli katkıları olan engin bilgi ve tecrübelerinden yararlandığım selef danışman hocam Dr. Öğr. Üyesi M. Akif NACAR 'a, bu çalışmanın nihayete ermesinde ciddi katkısı ve emeği olan danışman hocam Dr. Öğr. Üyesi M. Emin TENEKECİ 'ye ve çalışmam süresince desteğini esirgemeyen eşime teşekkür ederim.

ŞEKİLLER DİZİNİ

Şekil 1.1. Yıllara göre veri miktarındaki değişim 2
Şekil 1.2. GPU bant genişliği kapasitesinin yıllara göre değişimi 9
Şekil 1.3. CUDA işlem akışı 10
Şekil 1.4. Spark ekosistemi 13
Şekil 1.5. Spark SQL mimarisi 14
Şekil 3.1. Çoklu GPU iş istasyonu 30
Şekil 3.2. Satır ve Sütun yönelimli veri tabanları 34
Şekil 3.3. Spark SQL ve Spark DataFrame Python 1. Sorgu 36
Şekil 3.4. Spark SQL ve Spark DataFrame Python 3. Sorgu 36
Şekil 3.5. Apache Spark sorgu iş akışı 37
Şekil 3.6. SQL ve cuDF ile DataFrame Python 1. Sorgu 38
Şekil 3.7. SQL ve cuDF ile DataFrame Python 4. Sorgu 38
Şekil 3.8. GPU sorgu iş akışı 39
Şekil 4.1. Sorgu 1 performans karşılaştırma grafiği 40
Şekil 4.2. Sorgu 2 performans karşılaştırma grafiği 41
Şekil 4.3. Sorgu 3 performans karşılaştırma grafiği 42
Şekil 4.4. Sorgu 4 performans karşılaştırma grafiği 42
Şekil 4.5. Sorgu 5 performans karşılaştırma grafiği 43
Şekil 4.6. Sorgu 6 performans karşılaştırma grafiği 44
Şekil 4.7. Sorgu 7 performans karşılaştırma grafiği 44

ÇİZELGELER DİZİNİ

Çizelge 1.1. 2019 yılına ilişkin ortalama günlük veri miktarı 3
Çizelge 1.2. Spark Scala MapReduce işlemi örneği 12
Çizelge 1.3. cuDF ile parquet dosyasının işlenmesi 16
Çizelge 1.4. BlazingSQL ile parquet dosyasının işlenmesi 17
Çizelge 3.1. Tesla T4 GPGPU özellikleri 31
Çizelge 3.2. CPU özellikleri 31
Çizelge 3.3. Kullanılan yazılım kütüphaneleri sürümleri 32
Çizelge 3.4. Veri tabanında yer alan alanlar 33
Çizelge 3.5. CSV dosyasının Parquet dosyasına dönüştürülmesi 34
Çizelge 3.6. Sorgu türleri 35
Çizelge 3.7. Sorgu cümleleri 35

SİMGELER ve KISALTMALAR DİZİNİ

API	Application Program Interface(Uygulama Programlama Ara Yüzü)
CPU	Central Processing Unit(Merkezi İşlem Birimi)
CUDA	Compute Unified Device Architecture (Paralel Programlama Platformu)
ETL	Extract, Transform and Load
GPU	Graphics Processing Unit(Grafik İşlem Birimi)
GPGPU	General Purpose Graphics Processing Unit
HPC	High-Performance Computing (Yüksek Başarımlı Hesaplama)
RAM	Random Access Memory (Rasgele Erişimli Bellek)
RDD	Resilient Distributed Datasets (Esnek Dağıtık Veri Kümeleri)

1. GİRİŞ

Veri kavramı içinde bulunduğumuz yüzyılda önemi büyüdükçe artan bir unsur haline gelmiştir. Sağlık, ekonomi, sanayi, teknoloji, güvenlik vb. hemen hemen her alanda verilerden yararlanılarak geleceğe ilişkin nasıl yol alınacağı ve ne tür önlemler alınması gerektiğine yönelik anlamlı çıkarımlarda bulunulması sağlanmaktadır. Verilerin anlamlandırılması ile aşağıda belirtilen durumlar gibi birçok alanda önemli sonuçlar elde edilmektedir.

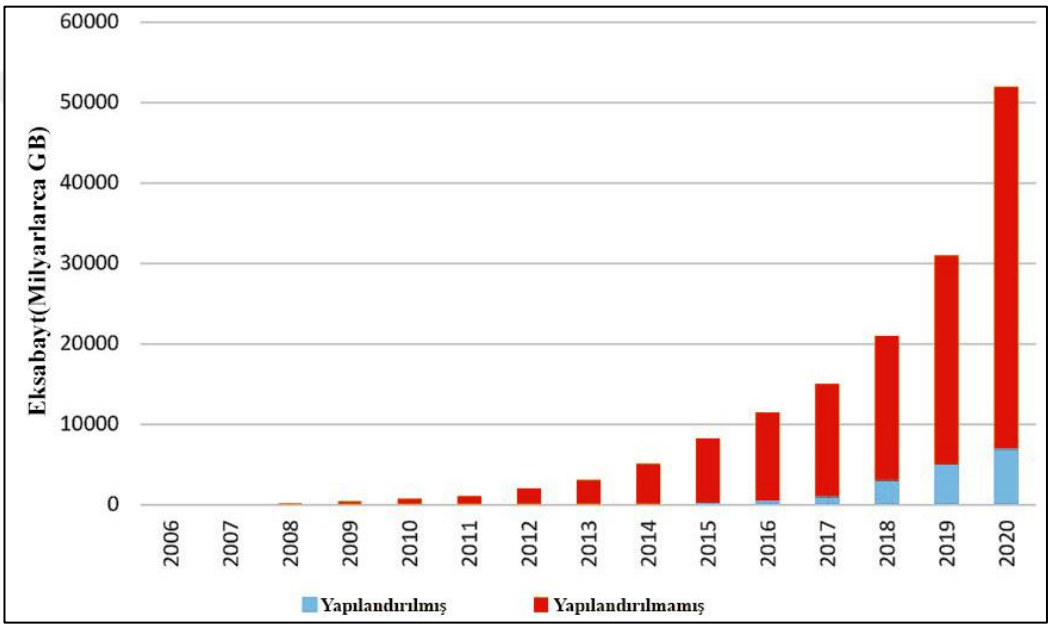
- İnsanların yaşam kalitelerinin arttırılmasına yönelik eylemler ve önlemler alınması
- Düzenli ve doğru tutulan bir veri ile daha tartışmasız çıkarımlarda bulunulması
- Bir problemin çözümüne yönelik stratejilerin başarılı olabilmesi ve etkisinin ölçülmesi
- Kaynakların daha etkili bir şekilde kullanılması
- Herhangi bir iş alanındaki memnuniyetin arttırılmasına yönelik stratejilerin belirlenmesi
- Dijital sistemlerde meydana gelen hataların azaltılmasına yönelik önlemlerin alınması
- Dijital sistemlerdeki zafiyetlere ilişkin önlem alınması

Veri, malumat veya bilgi birbiriyle benzer kavramlar olarak ilişkilendirilebilmektedir, fakat her birinin diğerine göre farklı rolleri ve anlamları vardır. Veri kavramı yaygın olarak, bilimsel araştırmalar ile ilişkilendirilse de işletmeler, kamu kurumları ve sivil toplum kuruluşları gibi çok sayıda kurum ve kuruluş, tarafından toplanmaktadır. Veriler ölçülür, toplanır, raporlanır ve analiz edilerek grafikler, görüntüler ve diğer analiz araçları kullanılarak görselleştirilebilirler.

Yakın zamana kadar veriler sınırlı olarak tutulmaktaydı. Hayatın hemen her alanında gerçekleşen dijital dönüşüm beraberinde daha fazla hacim ve daha fazla değişkenin olduğu büyük veriler ile yeni sistemlere olan ihtiyacın ortaya çıkmasına neden oldu. Geçmişte yaygın olarak kullanılan bilgisayar ile yürütülen çalışmalar da

kayda değer bir sorun olmazken, büyüyen veri miktarı ve bu verilerden daha çabuk sonuç alınması ihtiyacı farklı teknolojilerin ortaya çıkmasını sağlamıştır.

Verinin artış hızı, çeşitliliği ve büyümesine paralel olarak geleceğe dair öngörüler dikkate alındığında verilerin belirli bir yapıya göre organize edilmesi için çalışmalar başladı (Şekil 1.1.) ve veriler yapılandırılmaya başlandı (Rizatti, 2016). Schema.org topluluğunun (Microsoft, Google, Yahoo ve Yandex vd.) internetteki verilerin oluşturulması ve sürdürülmesi için yaptıkları çalışmalar neticesinde arama motorlarının daha anlamlı ve doğru sonuç vermesi sağlandı.



Şekil 1.1. Yıllara göre veri miktarındaki değişim

Verinin sınıflandırılması ve belirli formatlarda sunulması ile veri temelli işlemlerde daha detaylı ve kesin sonuçlar elde etmek, mantıklı karşılaştırmalar ve analizler yapılabilmesi sağlanmıştır. Veri miktarındaki devasa artış bu alanda yapılan çalışmaları da doğrudan etkilemiş ve bu artışa paralel olarak yeni aygıtlar ve sistemlerin ortaya çıkmasına neden olmuştur.

Verilerin katlanarak büyümesine ve veri üzerinde yapılan işlemlerin uzun sürmesine karşılık başlangıçta daha hızlı işlem yapabilen CPU'lar kullanılmaktaydı. Eğer mevcut sistem performans açısından ihtiyaç duyulan kapasitede değilse, tek yapılacak şey daha yeni ve daha yüksek kapasiteli bir işlemci alınarak soruna çözüm

bulunmaya çalışılıyordu. Bir süre sonra yonga-imalat endüstrisindeki gelişmeler belirli bir doygunluğa ve sınıra ulaştı, ama aynı zamanda veriler hızlı bir şekilde artmaya ve çok farklı alanlarda kullanılmaya başlandı.

Sadece günlük hayatın işleyişine bağlı olarak oluşturulan veriye (Çizelge 1.1.) bağlı olarak, bu hizmetin sağlanmasında rol oynayan internet sağlayıcıları, sunucular bulunduğu veri merkezleri, bu merkezlerde çalışan sistemler, çalışanlar uygulamalar vb. birçok bileşen bulunmakta ve bu sistemlerden dolayı ayrıca aktivite kayıtları(log) tutulmaktadır.

Çizelge 1.1. 2019 yılına ilişkin ortalama günlük veri miktarı

Veri Türü	**Miktarı**
Tweet	500 Milyon Adet
E-Posta	290 Milyar Adet
Facebook'ta oluşan veri	4096 Terabayt
Whatsapp	65 Milyon Adet
Arama motorları	5 Milyar Sorgu

Verilerdeki bu devasa artışa paralel olarak teknoloji ve sektördeki gelişmeler bir avantaj olmaktan ziyade bir gereklilik olarak ön plana çıkmaktadır. Günümüzde benzer alanlarda çalışan sektörlerin birbiriyle rekabet edebilmesinin ve avantajlı olabilmesinin yolu verinin gerçek zamanlı erişilebilmesi ve daha hızlı işlem yapabilmesine bağlıdır (Einav ve Levin, 2013).

Verilerin depolanması, işlenmesi, analiz edilmesi, çözümlenmesi ve görselleştirilmesi vb. birçok yönden performansın artırılmasına yönelik yeni çözümlerin bulunması önem taşımaktadır. Bununla birlikte büyük verilerin analizi ve bu analizlere ilişkin sonucun değerli bir meta haline gelmesi bu alanda da önemli gelişmelerin olmasını sağlamıştır (Mishra ve Sharma, 2015). MapReduce, Hadoop, Impala ve Spark büyük veri için geliştirilen yapılardan sadece birkaç tanesidir. Yatay ölçeklenebilirlik sayesinde büyük veriler işlenirken sunucunun yetersiz kalınması durumunda yeni bir sunucu daha eklenip verilerin daha verimli bir şekilde işlenmesi sağlanmaktadır (Faircloth, 2017). Twitter, Yahoo, Facebook, LiknedIn ve Google gibi büyük veriler ile çalışan şirketlerde de yine benzer çözümler kullanılmaktadır.

Büyük veri işlemede yaygın olarak kullanılan ölçeklenebilir yapılardan biri de Apache Spark'tır. Spark ile binlerce düğüme ölçeklendirilebilen sistem diğer büyük veri işleme yapılarına göre daha hızlı çıktı vermesi açısından tercih edilmektedir. Apache Spark'ın ön plana çıkmasının en önemli nedeni bellek içi hesaplama yapabilme özelliğinden dolayıdır (McDonald, 2018). RAM bellek kapasitelerinin artması ile birlikte disk erişimi ile yapılan işlemler yerine belleğe erişilerek performans artışının sağlanması geliştiriciler için bu alanda yeni fırsatlar ortaya çıkarmıştır (Drakos, 2019).

CPU'daki değişikliklerin geçmişe oranla daha durağan seyretmesi GPU gibi farklı alternatiflere yönelimi de beraberinde getirmiştir. Son yıllarda GPU teknolojisinde kayda değer bir ilerleme sağlandı. Modern GPU'lar sadece güçlü bir grafik kartı olarak değil, ileri derecede performans sağlayan paralel hesaplama özelliği ve yüksek bellek bant genişliği ile CPU'ya oranla çok daha hızlı işlem yapabilme kapasitesi açısından da tercih edilmeye başlandı (Şahin ve Külür, 2016). GPU'nun programlanabilir olması ve karmaşık problemlerdeki yüksek hesaplama başarımı nedeniyle günden güne daha çok ön plana çıkmasını sağlamıştır. GPU'nun hesaplamalardaki yüksek başarımının arttırılmasına yönelik çabalar GPU Hesaplama kavramının yaygınlaşmasını beraberinde getirmiştir. GPU'lar hesaplamalardaki yüksek kapasiteleri nedeniyle geleneksel mikroişlemcilere kıyasla iyi bir alternatif olarak ön plana çıkmaktadır. Sadece grafik işlemleri ve hesaplamalar için değil herhangi bir algoritma da GPU üzerinde koşturulabilmektedir. GPU'ların en önemli özellikleri; büyük ölçüde paralelleşebilme, yüzlerce çekirdek, binlerce thread ve CUDA ile programlanabilirlik olarak ifade edilmektedir (NVIDIA, 2016).

Paralelleştirme teknolojisi bir hesaplama özelliği olmakla birlikte geliştirilen mikroişlemcilerde sadece thread performansının artırımına yönelik geliştirmeler değil çekirdek sayısının da artırılması için çalışmalar yapılmaktadır. GPU'nun performansı ve potansiyeli gelecekteki bilgisayar sistemleri için büyük bir umut vadetmesinin yanında, GPU'nun mimarisi ve programlama modeli diğer tek tip işlemcilerden belirgin bir şekilde farklılığını yansıtmaktadır. Programlanabilen işlemci çekirdek sayısının artması toplam sistem çıktısının artmasını sağlamaktadır (Wolf, 2014). Aynı

zamanda işlemcilerin birleştirilmesi çok etkili bir şekilde yük dengelenmesini sağlar, çünkü herhangi bir işlem fonksiyonu tüm işlemci dizisini kullanabilecektir.

GPU'nun paralelleştirme özelliği veri tabanındaki iş yükünün en yüksek performans ile karşılanması için GPU'nun maksimum bellek bant genişliği ile birlikte programın tamamıyla paralel olacak şekilde tasarlanmalıdır. CUDA'nın ölçeklenebilir programlama modeli, paralel hesaplama uygulamalarının çok sayıda iş yükünün kaldırabilmesini sağlarken ve uygulamaya paralel olarak paralel işlemci çekirdeklerinin de ölçeklenmesini sağlar (NVIDIA, 2011a). GPU'dan elde edilen başarım birden fazla CPU'nun bir arada kullanılması ile sağlanabilir, fakat performans/güç tüketimi oranı açısından GPU kadar başarılı olamamaktadır. GPU'lar yaptıkları işe göre güç tüketimi açısından en verimli seçenektir (NVIDIA, 2011b). Basit organize edilmiş GPU uygulamaları, GPU kapasitesinin verimli bir şekilde kullanamamasından dolayı tükettiği enerji miktarı ve çok çekirdekli yapısı açısından etkili bir sonuç vermeyecektir.

1.1. Büyük Veri

Büyük veri, yapılandırılmış veya yapılandırılmamış muazzam büyüklük ve çeşitlilikteki veriyi tanımlamak için kullanılmaktadır (SAS, 2016). Önemli olan verinin miktarı değil, organizasyonların bu verileri nasıl ve ne amaçla kullandıklarıdır. Büyük veriler, daha iyi kararların alınması ve stratejik iş hareketlerine yol açan öngörüler için bu verilerin analizi belirleyici olmaktadır. Bu açıdan, büyük veri uygulamalarının analiz ve yürütme sürecinde olabildiğince doğru kullanması büyük önem taşımaktadır.

Büyük veri denildiğinde sadece verinin kendisini değil, bu alandaki teknolojiler, yöntemler ve sorunlardan da bahsedilmektedir. Veri miktarının hızlı bir şekilde artması bu kavramın farklı algılanmasına neden olmuş ve veri bilimcilerin bu alanda yapılacak çalışmalara yönelmesine ve bu alanın gelişmesine katkıda bulunmasını sağlamıştır. Büyük hacimlere sahip olan veri kümelerinin çeşit çeşit ve karmaşık yapıya sahip olmasından dolayı geleneksel yöntemlerin kullanılarak işlenmesi pek mümkün değildir (Oussous, 2017). Bu nedenle verilerin depolanması, işlenmesi,

analiz edilmesi ve görsel olarak anlamlandırılabilmesi için yeni teknikler ve teknolojiler ortaya çıkmaktadır.

Büyük veri miktarındaki artışa paralel olarak bu verilere erişim gereksinimi de artmaktadır. Sürekli olarak artan veri akışı ve bu verilerin işlenmesi özellikle bilgi çıkarımı açısından zordur. Bu nedenle, veri madenciliği, metin madenciliği gibi otomatik bilgi çıkarım yöntemleri ortaya çıkmıştır. Bunlar ekonomi, politika ve pazarlamadaki radikal dönüşümlerin arkasındaki süreçlerden bazılarıdır. Piyasada kullanılan birbirine bağlı yeni nesnelerin ortaya çıkmasından dolayı mevcut veri miktarı da büyük ölçüde artacaktır. Büyük veri sistemlerinde bulunan ana bileşenler şu şekilde ifade edilebilir (Avci Salma ve ark., 2017):

1. Veri
2. Veri Yönetimi
3. Verini İşlenmesi
4. Veri depolama ve sorgulama sistemi
5. Veri analiz sistemi
6. Kullanıcı ara yüzü ve görselleştirme sistemi

Büyük veri için başlangıçta hacim(volume), çeşitlilik(variety) ve hız(velocity) gibi ifadeler kullanılmaktaydı. Günümüzde ise, büyük veriler için 40'ı aşan özellik tanımlanmaktadır. Aşağıda büyük veriler için öne çıkan bazı özellikler yer almaktadır.

1.1.1. Hacim

Muhtemelen büyük veriler için en bilinen özellik hacimdir. Verinin boyutunu ifade etmektedir (Hoy, 2014). Büyük veri çözümlerine olan ihtiyaç verinin hacmi ile orantılı olarak artmaktadır. Günümüzde zettabayt ile ifade edilen veri boyutu yakında yottabayt ile ifade edilecektir.

1.1.2. Çeşitlilik

Çeşitlilik, sensörler, GPS cihazları, telefonlar ve sosyal ağlar gibi farklı farklı kaynaklardan elde edilen verileri ifade eder. Büyük veri söz konusu olduğunda, yalnızca yapılandırılmış veriler değil, aynı zamanda yarı yapılandırılmış ile çoğu

zaman hiç yapılandırılmamış verileri de kullanmak zorundayız. Yapısal olmayan veriler tanımlı bir veri modeline sahip olmayan veriler için kullanılmaktadır. Kısaca verinin heterojen yapısına ifade eder (Hoy, 2014). Büyük veriler ses, görüntü, video dosyaları, günlük dosyaları, makine ve sensör verileri, tıklama verileri gibi çoğunlukla yapılandırılmamış veriler olarak karşımıza çıkmaktadır.

1.1.3. Hız

Verilerin üretilmesi, oluşturulması, yenilenmesi vb. özellikler ile verinin aktığı hızı da ifade etmek için kullanılmaktadır (Hoy, 2014). Verinin üretilmesindeki hız aynı zamanda işlem sayısının ve çeşitliliğinin de artması sonucunu doğurmaktadır. Facebook günlük 4 petabayt veri üretildiğini, Google ise günde 3.5 milyardan fazla sorgu gerçekleştirildiğini belirtmektedirler.

1.1.4. Güvenirlilik

Büyük verilerin en şanssız özelliklerinden birisidir. Verinin hacmi, çeşitliliği, hızı gibi özellikleri arttıkça verideki doğruluğu azalmaktadır. Veri kaynağının kanıtlanması ya da güvenilirliği, içeriği ve buna dayanan analizin ne kadar anlamlı olduğu güvenirlilik olarak ifade edilir (Shafer, 2017). Sistemde bulunan gerçek olmayan veya yanlış verilerin sistemde birikmesinin önlenmesini gerektirir.

1.1.5. Değişkenlik

Verideki tutarsızlıkların sayısı ile farklı veri türleri ve veri kaynağından kaynaklanan çok sayıda veri boyutu nedeniyle veriler değişkendir. Anlamlı analizler tutarsız verinin anormallik ve aykırılık tespit yöntemleriyle bulunması ile sağlanır. Değişkenlik ayrıca, veri tabanına büyük verilerin yüklendiği tutarsız hız olarak da ifade edilmektedir (Shafer, 2017).

1.1.6. Geçerlilik

Verilerin kullanım amacı için ne ölçüde doğru ve kesin olduğunu belirtir (Shafer, 2017). Araştırmacılar verileri kullanmaya başlamadan önce zamanlarının büyük çoğunluğunu verilerin ayıklanması için harcamaktadır. Büyük veri analizinden elde edilecek avantaj bu verilerin esas amaç için ne kadar uygun olduğuna bağlıdır.

1.1.7. Görselleştirme

Büyük verilerdeki önemli özelliklerden birisi de görselleştirmedir. Mevcut görselleştirme araçları, bellek içi teknolojideki kısıtlamalar, zayıf ölçeklenebilirlik, işlevsellik ve yanıt süresinin uzun olması nedeniyle büyük veriler için teknik zorlukları beraberinde getirmektedir. Bir milyar veri noktasını çizmeye çalışırken geleneksel grafikler çok güvenilir değildir (Anaconda, 2016). Bu işlem için veri kümelemesi veya ağaç haritaları, güneş patlamaları, paralel koordinatlar, dairesel ağ şemaları veya koni ağaçları gibi verileri temsil etmenin farklı yolları ile karmaşık verilerin daha iyi ve etkili bir şekilde anlaşılması sağlanır.

1.1.8. Değer

Son ve tartışmasız olarak en önemlisi ise verinin değeridir. Eğer verilerden herhangi bir işletme değeri elde edilmiyorsa bu verilerin bir anlamsızdır (Cano, 2014). Hedef kitlenin iyi anlaşılması, işlemlerin optimize edilmesi ve işletme ve makine performansının iyileştirilmesi gibi birçok açıdan büyük verilerden yararlanarak önemli değerler bulunabilir. Büyük verilerin filtrelenmesi ile elde edilen raporlar şirketlerin stratejilerinin belirlenmesinde önemli rol oynamaktadır (Khan ve ark., 2014).

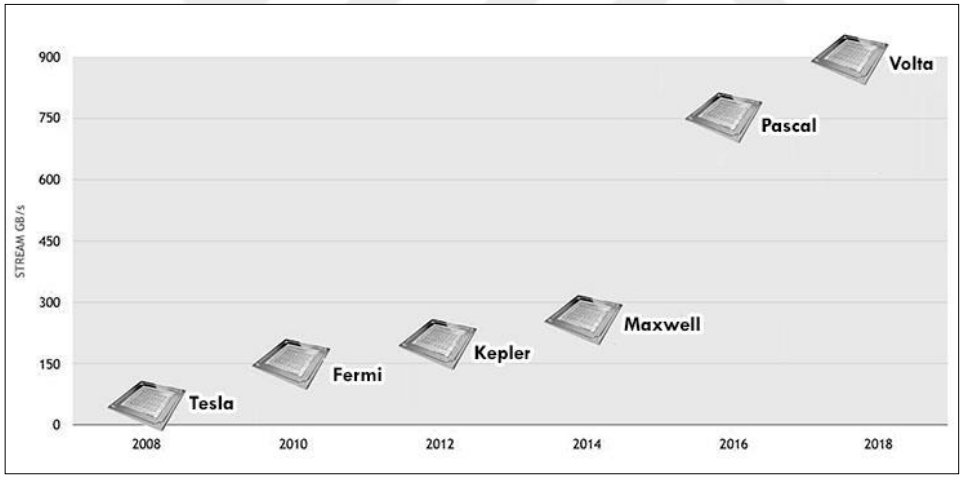
1.2. GPU

GPU, sadece grafik işlem birimi değil aynı zamanda programlanarak paralel hesaplama olanağı sunan, CPU'ya oranla daha fazla çekirdek ve bellek genişliği sağlamasıyla birlikte önemli ölçüde performansı etkileyen NVIDIA tarafından geliştirilen bir bilgisayar donanımıdır (NVIDIA, 2016a).

GPU'lar 1990'lı yıllarda ana CPU üzerindeki 2D ve 3D grafik işlem yükünü üstlenebilmesi için tasarlandılar. İlk zamanlarda çok fazla çekirdek barındırmayan GPU'lar daha sonra birden fazla pikselin paralel olarak işleyebildiler (Singer, 2013). GPU terimi resmi olarak NVIDIA firması tarafından GeForce 256'nın piyasa sürülmeye başlaması ile birlikte kullanıldı. Başlangıçtaki en büyük eksiklik GPU'nun kısıtlı olarak programlanabilmesiydi. Bu sorunun üstesinden gelebilmek için yeni nesil GPU'lar programlanabilirlik açısından daha iyi sonuçlar verdiler (Singer, 2013). Daha geniş ve ileri düzeyde programlanabilme trendi ile GPU'larda paralelleştirme

bakımından da ilerleme sağlandı. CUDA platformu ile başlangıçta C gibi dillerde programlama imkânı sunarken Fermi mimarisi ile birlikte C++ ve Fortran gibi dillerde de programlanabilir hale geldi. Fermi mimarisi ile GPU'lar HPC uygulamalarında da ihtiyaç açısından da memnuniyet verici sonuçlar veren ilk GPU mimarisidir (NVIDIA, 2016b).

Projelerde daha fazla kullanılmaya başlanan GPU, daha yüksek bellek kapasitesi olan ve daha gelişmiş paralelleştirme özelliği olan GPU'ların üretilmesini ve piyasaya sürülmesi sürecini doğrudan etkilemektedir. (Şekil 1.2.)'de görüldüğü gibi yeni nesil GPU'lar daha yüksek bant genişliğini sunmaktadır. CPU-GPU arasındaki ara bağlantı hızı GPU'ların gerçek kapasitesinin yansıtılmasında bir engeldir. Bu durumun üstesinden gelebilmek için AMD, CPU ve GPU'nun bir arada olduğu işlemciler üretti (AMD, 2016), fakat bu işlemcilerde de bellek bant genişliği hızı bakımından yeterli seviyeye ulaşılamadı.



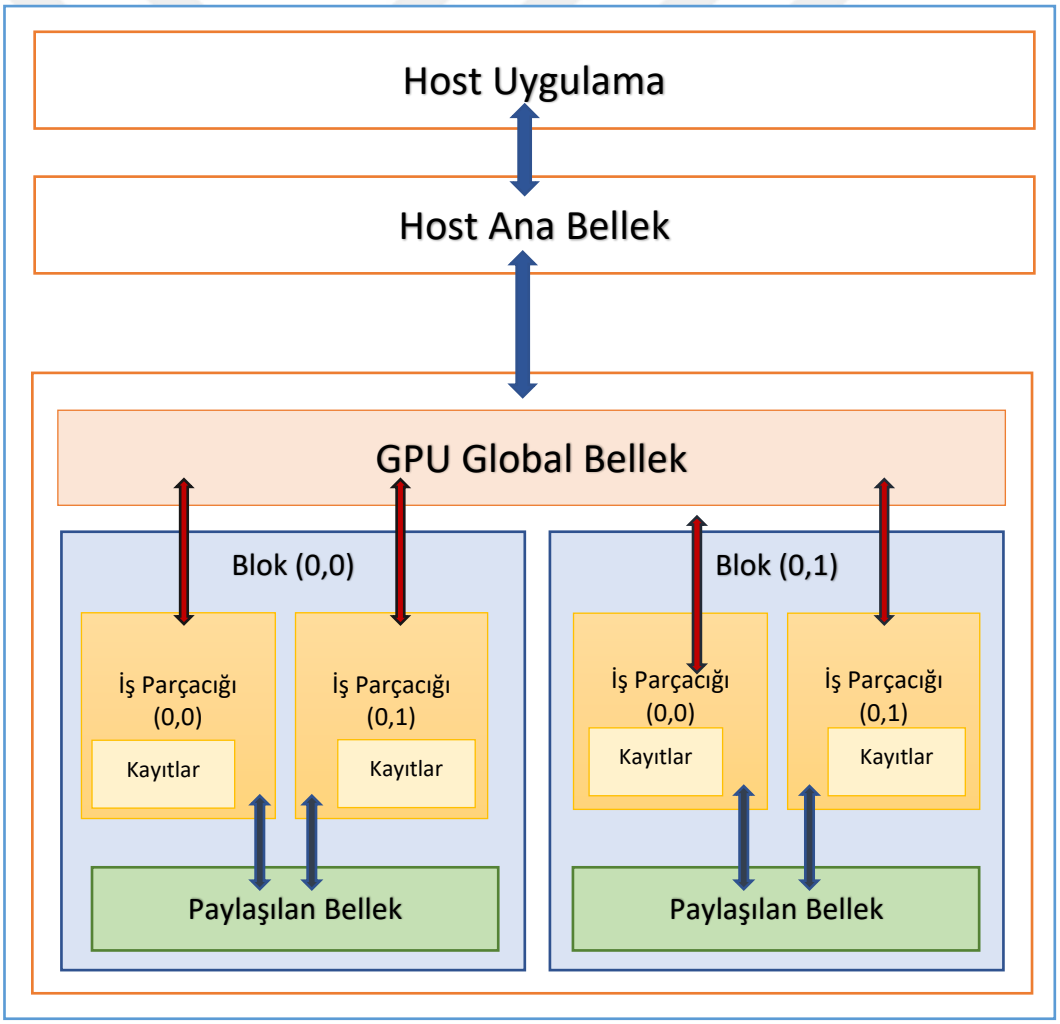
Şekil 1.2. GPU bant genişliği kapasitesinin yıllara göre değişimi

1.2.1. CUDA mimarisi

GPU, paralelleştirme işlemi için CUDA hesaplama mimarisini kullanır. CUDA ile birbiri ile veri paylaşımı yapabilen paralel dizilime sahip çekirdekler, CPU'nun tek düzen şeklinde yapacağı işi yayarak gerçekleştirir CUDA programlama modeli CPU ve GPU'nun birlikte kullanıldığı heterojen bir modeldir(NVIDIA, 2011a). CUDA uygulamalarında host ve device terimleri çok sık kullanılmaktadır. Host ibaresi ile CPU ve belleği ifade edilirken, device ile GPU ve belleği ifade edilir. Host üzerinde

yürütülen kodlar hem host hem de device belleğini yönetebilir ve device fonksiyonlarını başlatır (Harris, 2017).

GPU üzerinde gerçekleşen Şekil 1.3.'te belirtilen CUDA iş akışında uygulama aracılığıyla ana bellekten GPU global belleğine gönderilir. GPU aygıtındaki iş parçacıkları bloklar halinde gruplandırılmıştır ve GPU global belleğindeki veriye ulaşabilmektedirler. Her blok içinde iş parçacıkları ile paylaşılan bellek bulunmaktadır. Çalışma iş parçacıkları paylaşılan belleği global belleğe oranla daha hızlı kullanırlar. GPU'da işlemler yürütüldükten sonra sonuçlar uygulama aracılığıyla Host Ana Belleğine kopyalanır (Paz ve Plaza, 2011).



Şekil 1.3. CUDA işlem akışı

GPU iş parçacıkları tek boyutlu, iki boyutlu veya üç boyutlu olarak benzersiz iş parçacığı indeksi ile tanımlanmaktadır. GPU işlemcisinin yönetebileceği maksimum iş parçacığı sayısı GPU aygıtının donanım özelliklerine bağlı olarak değişkenlik gösterdiğinden azami bir sınırı vardır (Wincent, 2017). Daha fazla iş parçacığına ihtiyaç duyulduğunda host programı iş parçacıklarını gruplandırarak bloklara ayırmalıdır. Her blok tek, iki veya üç boyutlu ve benzersiz şekilde tanımlanır (Şekil 1.3.).

CUDA programlama modelinde CPU-GPU arasındaki bağlantı hızı nedeniyle yaşanan performans kaybının önlenebilmesi için bu çalışmadaki genel yaklaşım GPU'dan maksimum düzeyde faydalanarak işlemlerin daha hızlı gerçekleştirmesi öngörülmektedir. Bu amaçla veri tabanına ilişkin veri kümeleri GPU belleğinde tutularak CPU ve GPU arasında yaşanan veri transferinden kaynaklanan kaybın minimize edilmesi sağlanmalıdır.

GPU mimarisi temelde CPU'lardan farklıdır ve GPU'lar verimlilik için optimize edilmişlerdir (Wang ve Khan, 2015). CPU'da iyi sonuç veren bir algoritma GPU'da iyi çalışmayabilir. Bu nedenle, eski bir CPU uygulamasının yeniden derlenmesi performans açısından beklenen sonucu veremeyebilir. GPU'nun çalışma prensibi dikkate alınarak ve yeni algoritmalar ile yeni veri yönetim teknikleri uygulanmalıdır.

1.3. Apache Spark

Büyük veri kümeleri üzerinde çalışan ve ölçeklenebilir olarak tasarlanmış hata toleransı olan, kullanımı kolay ve hızlı dağıtılmış bir hesaplama yapısıdır. Apache Spark hata toleransı ve ölçeklenebilirliği sağlamak için sıkı bir programlama modeli kullanmaktadır(Apache). Spark üzerinde gerçekleştirilen herhangi bir hesaplama için değişmeyen veri kümeleri üzerinde belirli operatörleri kullanmaktadır (Zaharia, 2010). Başlatılan bir işlemin herhangi bir bölümünde problem çıktığında, Spark işlem yaparken oluşturduğu bir önceki denetim noktasına geri dönerek kaldığı yerden devam ederek işlemi tamamlamaya çalışır (Zaharia ve ark., 2012). Spark popüler yazım dilleri olan Python, Scala, Java, SQL ve R gibi diller ile çalışma imkânı sunmasından dolayı kullanımı kolaydır (Tutorials Point).

Spark açık kaynak olması, birçok yazılım diline bütünleşmiş işleme motorlarına sahip API'lerin olması ve geniş kütüphanelere sahip olmasından dolayı yaygın olarak kullanılmaktadır. Spark'ın sistem kullanımı açısından daha yüksek performansın sağlanabilmesine yönelik çalışmalar devam etmektedir. Makine öğrenme kitaplığını sürekli geliştirmekte ve harici veri kaynaklarına da erişilebilmesi için birçok barındırmaktadır (Apache).

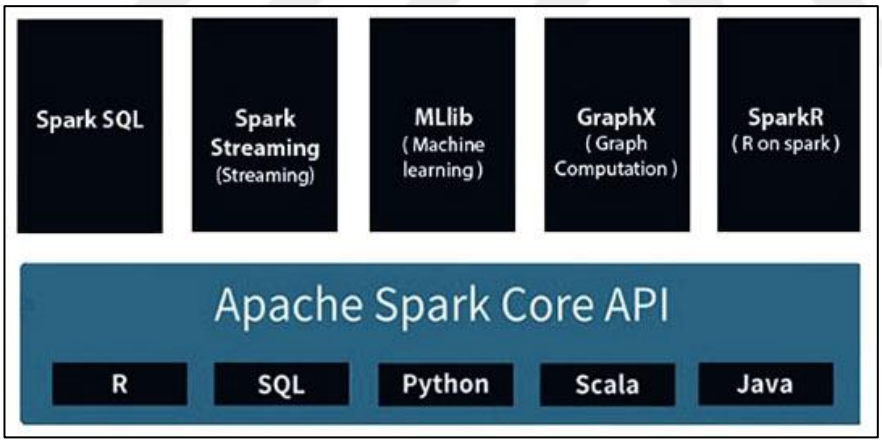
Spark makine öğrenmesi algoritmalarının ve veri dönüşüm mantığını paralelleştirerek yazılabilmesini sağladığından dolayı değişken boyut ve türdeki dağıtık depolama sistemleri üzerinde hızlı işlem yapabilmektedir (Meng ve ark., 2015). Spark başlangıçta Hadoop üzerinde yürütülen hesaplamaların hızlandırılması amacıyla orta çıkmıştır. Spark, depolama(storage) amacıyla Hadoop'tan yararlansa da Hadoop'un işleme(processing) işlevini kullanabildiği gibi kendi daha çok küme yönetimi hesaplaması ile de işlem yapmaktadır (Zahari ve ark., 2012). Spark, Hadoop'a bir alternatif olarak değil daha çok Hadoop ile yapılan işlemlerin performans açısından daha etkili bir şekilde gerçekleşmesini sağlamaktadır. Hadoop'un HDFS dosya yapısını kullanmasından dolayı Hadoop'tan bağımsız olduğu düşünülemez.

Spark uygulamaları Hadoop kümeleri üzerindeki bellek içi işlemlerde yaklaşık olarak 100 kat daha hızlı, disk üzerinde ise 10 kat daha hızlı işlem yapabilmektedir. Spark'ın bu yüksek performansı, disk üzerindeki okuma/yazma yerine bellek kapasitesini etkili bir şekilde kullanabilmesinin sonucudur (Sinha, 2019). MapReduce özelliği Spark'ın bağımsız bir şekilde işlemlerinin uygulanabilmesini sağlar (Çizelge 1.2.).
R ve Python dillerindeki DataFrame'ler ile alan adları ile verilerin sınıflandırılmasını ve düzenini sağlama imkânı sunmaktadır (Apache).

Çizelge 1.2. Spark Scala MapReduce işlemi örneği

```
val txtDosyasi = sparkContext.textFile("hdfs://...")
val  kayitSayisi = txtDosyasi.flatMap(line => line.split(" "))
            .map(kelime => (kelime, 1))
            .reduceByKey(_ + _)
kayitSayisi.saveAsTextFile("hdfs://...")
```

Spark Engine veya Spark Core ana yapıyı oluştururken, kütüphaneler ile farklı alanlara yönelik çözümler sunmaktadır (Şekil 1.4.). MLib ile dağıtılmış makine öğrenmesini de yine bellek-temelli yaklaşım ile yürütmektedir ve kümeleme, sınıflandırma ve regresyon gibi makine öğrenmesi algoritmalarını barındırmaktadır. GraphX ise Spark'ın dağıtılmış grafik-işleme kütüphanesidir ve grafik analitiği görevlerini basitleştirir (Sinha, 2019). GraphX, ETL (Extract, Transform & Load) işlemini, keşif analizini ve yinelemeli grafik hesaplamasını tek bir sistemde birleştirir(Dayananda, 2019a). Spark Streaming ile akış analitiğine ilişkin RDD dönüşümlerini sağlayarak verilerin gerçek zamanlı olarak analiz edilmesini sağlamaktadır. Spark SQL ile yapılandırılmış veya yarı-yapılandırılmış veriler üzerinde sorgular SQL sorgularının gerçekleştirilmesi gibi çözümleri barındırmaktadır(DATA FLAIR, 2017). Ayrıca, değiştirilmemiş Hadoop Hive sorgularının mevcut dağıtımlarda ve verilerde yaklaşık olarak 100 kata kadar daha hızlı çalışmasını ile birlikte Spark ekosisteminin geri kalanıyla güçlü bir entegrasyonu sağlamaktadır (Tutorials Point).



Şekil 1.4. Spark ekosistemi

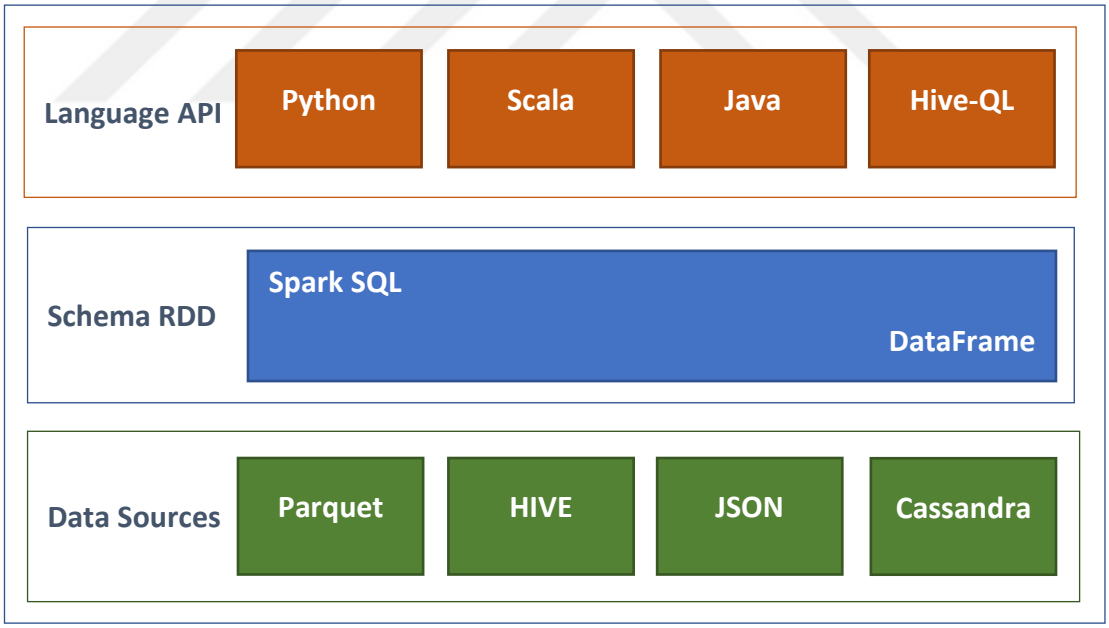
1.3.1. Spark SQL ve DataFrame

Spark SQL, yapılandırılmış verilerin işlenebilmesini sağlayan Spark bileşenidir ve dağıtılmış SQL sorgu motoru olarak işlev görmektedir. Spark SQL'in önce çıkan özellikleri şunlardır (Apache);

- SQL sorgularının Spark ile sorunsuz bir şekilde entegrasyonu ile yapılandırılmış veriler üzerinde sorgulama yapabilmemize sağlar.

- Scheme-RDD, Apache Hive, Parquet ve JSON dosyaları gibi çeşitli veri dosyaları üzerinde tek bir arayüz ile sorgulama yapma olanağı sağlar.
- Mevcut veri depoları için Hive sorgularının çalıştırılabilmesini sağlar.
- JDBC ve ODBC üzerinden bağlantı olanağı sağlar.
- İnteraktif ve uzun sorgular için aynı motoru kullanır ve hata toleransı için RDD model avantajlarından yararlanır.

Spark SQL Language API, Schema RDD ve Data Source olmak üzere üç farklı katmandan oluşur (Şekil 1.5.). Farklı programlama dilleri ile uyumludur. Spark Core RDD veri yapısı olarak adlandırılan özel bir veri yapısı ile tasarlanmıştır. Spark SQL şemalar, tablolar ve kayıtlar ile çalışabilir. Var olan bir kayıt dosyası için geçici RDD şeması tanımlanabilir. Spark Core metin dosyaları, Avro dosyaları vb. formatlardaki veri kaynaklarını kullanırken, Spark SQL farklı veri kaynaklarını kullanmaktadır. Bu veri kaynakları Parquet, JSON, Cassandra veri tabanı veya Hive tabloları olabilir (Tutorials Point).



Şekil 1.5. Spark SQL mimarisi

Spark SQL farklı veri kaynaklarıyla kolay uyum sağlamasından dolayı Spark ekosisteminde çok önemli bir yer tutmaktadır. Spark SQL diğer SQL veri tabanlarına alternatif olarak değil birçok işi yapabilen sorgu ara yüzü olarak ön plana çıkmaktadır

(Techvidvan Team, 2018). Spark'ın performansı ve sunduğu diğer avantajlara ek olarak GPU'nun işlem yapma mantığına uygun olan ve büyük veri işlemede önemli bir yeri olan sütun tabanlı veri tabanları ile kayda değer performans göstermesi nedeni ile bu çalışmada yer almaktadır.

Spark SQL ile birlikte DataFrame kavramı da öne çıkmaktadır. DataFrame, sütunlar halinde düzenlenen ve adlandırılan dağıtılmış bir veri koleksiyonudur. Kavramsal olarak, en uygun tekniklerle oluşturulmuş ilişkisel tablolara eşdeğerdir (McDonald, 2015). Bir DataFrame Hive dosyaları, yapılandırılmış veri dosyaları, harici veri tabanları veya RDD'ler gibi birçok kaynaktan oluşturulabilir. DataFrame'ler için genel özellikler şunlardır (Dayananda, 2019b);

- Avro, CSV, Cassandra gibi dosya formatları ile HDFS, Hive Tabloları MYSQL vb. depolama sistemlerini desteklerler.
- Bir kümden büyük kümelere ve kilobayt boyutundan petabayt boyutuna kadar olan verileri işleyebilirler.
- Spark Core ile bütün büyük veri kütüphaneleri ile kolayca entegre edilebilirler.
- Spark SQL Catalyst ile son derece optimize edilmiş kod oluşturulabilir.

Spark SQL ayrıca sorguları hızlı hale getirmek için maliyete dayalı bir optimize edici, sütunlu depolama ve kod oluşturma gibi özellikleri barındırır (Armbrust, 2015). Aynı zamanda, binlerce düğüme ölçeklenebilmesi ve tek uygulamada geliştiriciler açısından karmaşık olan sorgu ve analizlerin kolay bir şekilde gerçekleştirilmesini sağlamaktadır.

1.4. cuDF ve BlazingSQL

Python GPU DataFrame Kütüphanesi olan cuDF, verilerin birleştirilmesi, filtrelenmesi, toplanması gibi veri yönetimi işlemlerini GPU üzerinde gerçekleştiren bir DataFrame kütüphanesidir. Rapids tarafından sunulan açık kaynaklı yazılımdır. cuDF Apache Arrow sütunlu bellek formatına dayalı olarak geliştirilmiştir. Veri bilimcilerin yaygın olarak kullandığı Python ile kullanım kolaylığı sunmaktadır (RAPIDS). Çizelge 1.3.'te de Python ile kullanımının ne kadar kolay olduğu

görülmektedir. Veri bilimcileri için tanıdık olan Pandas benzeri bir API olarak verilerin GPU üzerinde yürütülmesini sağlayarak CUDA ile yazılım geliştirme detaylarına bağlı kalmadan iş akışlarının hızlandırılmasını amaçlamaktadır.

Çizelge 1.3. cuDF ile parquet dosyasının işlenmesi

```
…
df = cudf.read_parquet(‘/dosya-adresi.parquet)
for column in df.columns:
   print(df[column].mean())
…
```

cuDF, veri miktarının tek GPU belleği boyutunu aşmadığı durumlarda iyi bir seçenek olurken, belleği aşan veriler için iş akışının çoklu GPU'lar üzerinde yürütülebilmesi için geliştirilen Dask-cuDF' in tercih edilmesi daha başarılı sonuçlar verecektir. Dask DataFrame başlangıçta Pandas'ın birden fazla CPU üzerinde ölçeklendirilmesi ve uygulanmasını sağlamak için tasarlanmıştır (RAPIDS). cuDF kütüphanesinin kullanılabilmesi için CUDA'nın 9.2 veya daha yeni sürümü ile Tesla T4 GPU'su gerekmektedir.

BlazingSQL ise cuDF ile GPU üzerinde gerçekleştirilen ETL ham verilerinin DataFrame fonksiyonları yerine SQL ara yüzü ile veri kümelerinin cuDF'in GPU üzerinde yaptığı işlemleri yapma kolaylığı sunmaktadır(RAPIDS). cuDF'te olduğu BlazingSQL'de Python dili kullanım kolaylığı sunmaktadır(Çizelge 1.4.). Apache Parquet meta verilerini tutmasından dolayı BlazingSQL ile tabloların oluşturması daha kolay olmaktadır (Weiss, 2019). Tablo oluşturulduktan sonra ise SQL sorguları kolaylıkla gerçekleştirilmektedir. Sorgu sonrasında sonuçlar GPU belleğinde tutulduğu için bu sonuçlara bağlı olarak GPU üzerinde yeni işlemler ve sorgular yapılabilmektedir. BlazingSQL, SQL'in bütün fonksiyonlarını yerine getiremese de birçok SQL fonksiyonunu desteklemektedir. Çizelge 1.4. BlazingSQL'in Python dili ile kullanımına ilişkin örnek görülmektedir.

Çizelge 1.4. BlazingSQL ile parquet dosyasının işlenmesi

```
….
# Parquet dosyasından tablo oluşturulması
bc.create_table(deletions, '/content/deletions.parquet')

# SQL Sorgulama işlme
sonuc = bc.sql('SELECT count(*) FROM main.deletions GROUP BY creator(key)').get()
sonuc_gdf = sonuc.columns

#Sonuçların gösterilmesi
print(sonuc_gdf)
….
```

1.5. Problem Tanımı

Dijitalleşmenin hayatın her alanında yer almasına bağlı olarak teknolojinin kullanım alanları ve uygulanmasındaki yöntemlerde de farklılıklar ortaya çıkmıştır. Başlangıçta belirli fonksiyonları yerine getirmek için üretilen bir ürün daha sonra konforun arttırılması, güç tüketiminin optimize edilmesi, arızalarının azaltılması, hataların giderilmesi vb. birçok değişken dikkate alınarak rekabete ve geliştirilmeye dayalı bir sürecin parçası olmuştur. Dünya çapında yaygın bir ağ ve bu ağdaki nesnelerin belirlenmiş kurallara göre birbirileriyle iletişim içinde olmaları ve verilerin benzersiz bir şekilde adreslenmesi farklı yaklaşımların ortaya çıkmasını sağlamıştır. Bu gelişmelere paralel olarak internetin günlük yaşamın bir parçası olan her nesnede yer alması ve bu nesnelerin oluşturduğu aktivite kayıtlarının kullanıcının iznine bağlı olarak belirli merkezlere aktarılabilmesi neticesinde hizmet sağlayıcılarının bu verilere bağlı olarak geliştirme, güvenlik vb. birçok yönden ihtiyaca ve hedefe yönelik çalışmalar yapmaları mümkün olmuştur.

İnternet servis sağlayıcıları, uygulama sunucuları ve uygulamalar gibi teknolojinin kullanıldığı her alanda kullanıma ve çalışma şekline bağlı olarak veriler oluşmaktadır. Bu veriler kullanıcıları günlük aktiviteleri, uygulama çalışırken oluşan hatalar ve sistemlere yapılan saldırılar gibi farklı farklı aktivite kayıtları olarak karşımıza çıkmaktadır. İşletmeler, kamu kurumları ve sivil toplum kuruluşları da dâhil olmak üzere çok sayıda kurum ve kuruluş tarafından benzer veriler tutulmaktadır. Bu veriler tehditlerin tahmini ve önlenmesi, sosyal ve sağlık hizmetlerinde israf ve

hataların önlenmesi, vergi kaçakçılığının ve sahteciliğin önlenmesi, suç tahminleri ve önlenmesi ve müşteri memnuniyetinin artırılması gibi birçok alana büyük katkıları olmaktadır. Bu verilere ilişkin raporlar oluşturulurken grafikler, görüntüler ve diğer analiz araçları kullanılarak görselleştirmenin sağlanması ile anlamlı çıkarımlarda bulunulması daha kolay olacaktır.

Büyük veya küçük birçok işletmede bilgi işlem birimlerinde sistemlerde gerçekleşen aktivitelere ilişkin log izleme işlemi gerçekleşmektedir. İzleme işleminde amaç ağınızı potansiyel olarak hassas bilgilere güvenlik saldırıları ile erişmek üzere girilebilecek güvenlik açıklarının tespit edilerek önlem alınmasını sağlamaktır. Birçok sistemde her gün milyarlarca veri toplanmaktadır. Bu ham veriler neticesinde oluşan kayıtların analizi ve analitiğine ilişkin işlemlerin daha kısa sürede gerçekleştirilebilmesi ve ihtiyaca yönelik raporların oluşturulabilmesi ile bu verilerin kullanıldığı sektörlerde oluşacak kaybının minimize edilmesi büyük önem taşımaktadır. Bu durumda verilerin analizine ilişkin sürecin daha kısa sürede gerçekleşmesi büyük bir ihtiyaç haline gelmektedir.

Hacim, verinin oluşturulma hızı ve çeşitlilik vb. açılardan büyük olan bu veriler üzerinde daha hızlı ve doğru sonuçlar alınabilmesi için donanım kapasitesinin arttırılması, yazılımlardaki iyileştirmeler, yeni kütüphane veya yazılımların geliştirilmesi ve yeni donanımların ortaya çıkması gibi farklı çözümler ortaya çıkmıştır. Bu kapsamda, başlangıçta farklı amaçlar için ortaya çıkmış ve birçok alanda başarılı sonuçlar sergilemiş olan GPU aygıtı kullanılacaktır.

1.6. Çalışmanın Amacı

Bu çalışma ile bellek içi veri analitik işlemlerinde tam performansı engelleyen tıkanıklıkların GPU'nun sağladığı avantajların tümüyle kullanılması ile minimize edilerek performansa olan etkileri gözlemlenecektir.

Veri tabanlarının GPU üzerinde işlenmesi günümüzde birçok çalışma ve araştırmanın yapıldığı yeni bir alandır. GPU'nun programlanabilir olması büyük çıktısı olan kurum-şirket uygulamalarında CPU temelli uygulamalara göre daha yüksek performans gösterebilecek potansiyelde bir araç olarak görülmektedir. Bu

kapsamda log dosyalarının GPU'nun işlem gücü ve temel tasarım ilkeleri dikkate alınarak pratikte uygulanabilirliği ve verilerin en uygun şekilde analizi bakımından performansa katkısı ölçülecektir.

Veriler kullanıldığı alana göre farklılık gösterse de temelde aynı değişkenleri barındırırlar ve benzer özelliktedirler. Araştırmamıza ilişkin sonuçların karşılaştırılabilmesi için büyük veriler üzerinde etkili işlem yapabilen Apache Spark kullanılarak gerçekleştirilen sorgular GPU üzerinde de gerçekleştirilerek, GPU üzerinde yürütülen sorguların gerçekleşme süreleri ile kıyaslanacaktır.

1.7. Çalışmanın Önemi

Günden güne gelişen dijital dünyanın katlanarak büyüyen ve artan çeşitliliğiyle birlikte, bu gelişimin bir yük olmaktan ziyade daha hızlı, daha etkili, daha makul çıkarımlarda bulunarak iş dünyasının ve organizasyonların ve başta kamu kurumların daha iyi hizmet verebilme imkânı sağlanmalıdır. Aksi halde, gelişme olumsuz beraberinde sadece problem getirmiş olacaktır. Hadoop, Apache Spark, GPU vb. teknolojiler bu problemlerin aşılması için ortaya çıkmış çözümlerdir.

Büyük ölçekli veri analitiği sistemlerinde genelde şebekelerde yaşanan erişim problemlerinden kaynaklanan zaman kaybı ile disk erişimindeki kayıpların giderilerek performansın arttırılması hedefleniyordu. Son yıllarda bu alanlarda büyük gelişmeler kaydedildi. Şebeke bağlantı hızı 10 Mbps'den 1 Gbps hatta 10 Gbps gibi erişim hızlarına ulaşıldı. Disk hızlarında ise yeni nesil SSD diskler ile 50 Mbps'den 2000 Mbps gibi hızlara erişildi. CPU çekirdek sayısı günümüz iş akışları bakımından Spark gibi bellek içi veri işleme sistemleri için yetersiz olmaktadır. Bellek içinde gerçekleştirilen sıralama sıkıştırma vb. işlemler CPU'nun tüm kapasitesini kullanmaktadır. CPU'nun bu iş yükünün azaltılması için farklı alternatifler bulunmaktadır.

İlk olarak CPU çekirdek sayısının arttırılması ile başlayan ve daha sonra CPU sayısının arttırılması ile yazılımların da çoklu çekirdek veya CPU'yu kullanabilecek şekilde tasarlanarak birlikte performans artışına yönelik çalışmalar gerçekleştirildi. Birden fazla CPU'nun bulunduğu sistemler ile birlikte yeni problemler ortaya

çıkmıştır. Bu problemlerden ilki enerjinin verimliliğidir. Çok çekirdekli CPU'lar, çoklu CPU'lar ile kıyaslandığında daha verimlidirler. Çekirdeklerin bir arada olması CPU'lara iletilen sinyallerin yönetiminin tek yonga ile sağlanması ve tek fan ile soğutulması nedeniyle çoklu CPU'ya oranla daha az enerji tüketirler. İkinci problem ise çekirdek sayısı veya CPU sayısının artmasına bağlı olarak ana kart üzerinde daha fazla yer kaplaması ve bu artışlara paralel olarak maliyetin de artmasıdır. Son ve en önemli problem ise çok çekirdekli CPU'lar aynı belleği kullanmalarından dolayı çoklu CPU'lara oranla daha hızlı çalışmaktadırlar. Elektrik sinyallerinin çekirdekler arasındaki mesafenin daha kısa olması nedeniyle daha hızlı olması çoklu işlemcilere göre performans açısından daha olumlu sonuçlar vermektedir.

Bir işlemcideki çekirdek sayısının artmasının bağlı olarak performansında önemli bir artış sağladığına değinmiştik. CPU'lar GPU'lara oranla daha düşük sayıda çekirdek barındırmaktadırlar. GPU'lar ise binlerce çekirdek ve yüksek bellek kapasitesine sahip olmasından ve programlanabilmesinden dolayı yüksek performans gerektiren çalışmalarda ve uygulamalarda kullanılmaktadırlar. Bu çalışma ile verilerin GPU'nun yapısına uygun olacak şekilde paralelleştirilerek büyük veri iş yükünün GPU üzerinde yürütülmesinin performans artışına katkıda bulunup-bulunmayacağına bağlı olarak bu alanda ihtiyaç duyulan donanım, maliyet ve performans ilişkisi için karşılaştırma yapabilmek mümkün olacaktır.

Bu çalışmada da iki farklı hesaplama teknolojisi olan Apache Spark ve GPU'nun sağladığı avantajlardan yararlanarak, büyük verilerde filtreleme, sorgulama ve hesaplama, işleme, inceleme vb. işlemlerin daha hızlı bir şekilde gerçekleştirilebilmesi için karşılaştırmalar yapılacaktır. Performansta gözlemlenecek farklılıklar sonucunda bilimsel çalışmalarda veya işletmelerde ihtiyaç duyulan kaynak kullanımında verim-performans-maliyet bakımından karar verme sürecinin daha kolay gerçekleşmesine katkıları olacaktır.

2. ÖNCEKİ ÇALIŞAMALAR

Modern GPU'ların hesaplama kapasitesinin çok hızlı bir şekilde geliştirilmesi ve bellek bant genişliğinin genişletilmesi sayesinde, paralel iletişim ve senkronizasyon, sürekli performans ölçeklendirmesi en çok ilgi duyulan konulardan biri haline gelmiştir. Bu özellikle yeni ortaya çıkan büyük veri uygulamaları için geçerlidir.

Düşüncelerimiz, işimiz ve yaşantımız gibi hayatın her alanındaki değişim, dönüşüm ve gelişimdeki, etkisinden dolayı veriler çok büyük bir değer olarak görülmelidir (Mishra ve Sharma, 2015). İçinde bulunduğumuz bilgi ve teknoloji çağının veri miktarının öngörülemez şekilde katlanarak artmasına neden olmaktadır. Verilerin analizi ile faaliyetlerine yön veren kurum ve kuruluşlar rekabet gücü açısından büyük bir avantaj yakalamaktadır (Opher ve ark., 2014).

Büyük veriler genellikle yapılandırılmamıştır ve daha fazla gerçek zamanlı analiz gerektirmektedirler. Bu durum veri toplama, aktarma, depolama ve büyük ölçekli veri işleme mekanizmaları için yeni sistem mimarilerinin oluşmasına neden olmaktadır. Hu ve ark. (2014), çalışmalarında uzman olmayan kişiler için genel bir resim sunmayı ve uzman olan kişiler için ise kendi büyük veri çözümlerini geliştirmelerini amaçlayan büyük veri analitiği platformları için bir kaynak taraması ve sistem eğitimi sunmaktadırlar.

Shamsuddin ve Hasan (2015), Büyük verilerin ortaya çıkışı ön görülemez büyüklükte büyük ölçekli örnekler sunmasının yanında daha karmaşık problemler ile karşılaşmamıza rağmen hesaplanması ve işlenmesi bakımından daha güçlü ve hızlı altyapıların ile iyi bir veri yönetim sistemine ihtiyaç duyulmaktadır. Geleneksel hesaplama modelleri ile kıyaslandığında günümüzdeki veri boyutu ve çeşitliliğine bağlı olarak veriye olan yaklaşım, verinin sunumu ve daha hızlı anlamlandırılması bir gereklilik haline gelmiştir.

Hu ve ark. (2014), büyük verilerin analitiği, güçlü destek platformları ile büyük veriler üzerinde henüz ortaya çıkmamış gizli modellere ve bilinmeyen ilgileşimler gibi potansiyelleri ortaya çıkarmak için analiz algoritmalarını çıkarma ve etkili

kullanabilme sürecidir. Bu bağlamda, büyük veriler için farklı yöntem ve yaklaşımların analiz sürecinde performans bakımından büyük önem taşıdığını göstermiştir.

Veriler genellikle ideal olarak biçimlendirilmese de, kuruluşlar mevcut yapılarını değerlendirmeli ve nerede olursa olsun bilim adamlarının deney, deneme ve diğer araştırmaların bilimsel analizini daha iyi işbirliği yapmasını ve bu sürecin hızlandırmasını sağlayan bir çözüm uygulama ihtiyacı duymalıdır. Araştırmacılar ve bilim insanlarının bu bilgi hazinesinde gezinmeleri ve istenen sonuçlara ulaşmaları zor olabilir. Bu büyük verilerin hacim, hız ve çeşitliliğinden kaynaklanmaktadır (Wang D., 2018).

Hoffman (2017), büyük verilerin hangi karakteristik özelliğinin boğulma etkisine neden olduğunu araştırdığı iki aşamalı tedarik zinciri modeli kullanılarak, verinin ‘hız ’,‘ hacim ’ve‘ çeşitlilik ’gibi büyük veri kolları bir simülasyona aktarılmıştır. Bu simülasyondan elde ettiği bulgular, verinin 'hız' karakteristiğinin göreceli olarak performansı artırmak için en büyük potansiyeli taşıdığını göstermektedir.

Kullanıcılar ve büyük veri analitiği arasındaki artan boşluk, büyük verinin hacmi, çeşitliliği ve hızının karşılaştığı zorlukları ele alan yenilikçi araçlar gerektirmektedir. Bu nedenle, bu kadar büyük veri hacmini analiz etmek hesaplama açısından yetersiz kalır. Ayrıca, büyük veri uygulamaları ve veri bilimi alanındaki gelişmeler, Yüksek Performanslı Bilgi İşlem çözümünün kilit bir konu haline geldiği ve son yıllarda dikkat çektiği ek zorluklar ortaya koymaktadır (Ahmed ve ark., 2018).

Weaver (2014), daha fazla bilgiyi daha küçük alanlarda saklayabilme becerimiz arttıkça, toplanan verinin değeri artık yalnızca koleksiyonun boyutuna bağlı değildir, aynı zamanda ondan ne kadar hızlı bilgi alabileceğimize de bağlıdır.

Büyük veri hacmi, heterojen veri, yüksek veri üretme ve güncelleme oranı, yüksek veri işleme gereksinimi ve geniş ölçüde ayrılmış veri kaynakları gibi veri mühendisliğinde veri yönetimi beraberinde bazı zorlukları da getirmektedir. Bu sorunların üstesinden gelinebilmesi için geleneksel veri yönetim teknolojileri yeterli olmamaktadır. Verinin

analizi, entegrasyonu, raporlanması ve veri kalitesinin değerlendirmesi de dâhil olmak üzere veri yönetiminin temel sorunları arasında yer almaktadır.

Williams ve ark. (2014), endüstriyel donanımların aktif olarak gözlemlenebilmesi için zamana bağlı olarak çok hızlı akan büyük ölçekli sensör verilerinin akış sırasında izlenmesi ve daha sonrada ihtiyaç duyulduğunda analiz edilebilmesi için bu verilerin saklanması gerekliliği bellek içi veri işleme yöntemleri başarılı çözümler olarak değerlendirmiştir.

Bellek içi hesaplama önemini yüksek performanslı hesaplamalı kimya deneylerinin hızlandırması, kısa vadeli enerji taleplerinin ortaya çıkarabilmesi için enerji tüketim modellerini tespit edilmesi ve ölçeklenebilir bir ara katman yazılımı mesaj kuyrukluma servisini yönetilebilmesi gibi çeşitli uygulamalardaki başarısıyla büyük oranda ispat etmiştir (Kendall ve ark., 2000).

HongJu ve ark. (2017), bilgi teknolojilerinin takibi ve yeni sistemlerin kullanılması ve özellikle büyük veri teknolojisindeki gelişmelere dikkate alınarak ile harp mühendisliğinin büyük ölçekli verilerin yönetilmesinden kaynaklanan zorlukların aşılarak, hızlı veri üretimi ve güncellemesi ve birim zamanda yüksek veri işlenmesi gerekliliğini belirtmiştir.

Wang ve Khan (2015) çalışmalarında, Apache Spark ile gerçekleştirilen belirli bir işteki performans, giriş verilerinin türü ve boyutu ile kullanılan algoritmaların tasarımı ve uygulanması, hesaplama yeteneğinin performans üzerindeki etkileri için tahminde bulunmanın ve zorlukların üstesinden gelinmesi için geliştirdiği tahmin modelinin yüksek derecede doğru sonuçlar vermiştir.

NVIDIA'nın 2006 yılında CUDA ile GPU üzerinde paralel hesaplama olanağı sunmasından beri farklı amaçlar için birçok çalışma yapılmada GPU temelli veri tabanı çalışmaları henüz yolun başında olsa da bu alanda ciddi çalışmalar devam etmektedir. Farooqui ve ark. (2016), entegre GPU sistemleri, verinin yoğun olarak kullanıldığı uygulamaları hızlandırmak için düşük maliyetli ve enerji açısından verimli bir seçenektir. GPU hesaplama yükünü azaltması ve başarılı kaynak planlaması için iyi bir seçenek olsa da, bazı zorlukları da bulunmaktadır. Bu zorlukların başında ise

yüksek performans elde edilebilmesi için iyi derecede uygulama bilgisi gereklidir. Wrede ve Ernsting (2016), değerlendirmeler göstermektedir ki GPU'lar belirli durumlarda kayda değer bir hızlanma sağlayabilir. Ancak, kullanılan donanım ve kullanıcı fonksiyonlarının karmaşıklığı bu sonucun sağlanmasında önemli rol oynamaktadır.

CUDA ve OpenCL'in tanıtılması ile birlikte GPU'ların programlanması çok daha verimli hale gelmiştir. Matematiksel olarak video işleme ve dönüştürme görevlerinin büyük ölçüde arttırmak için kullanılan GPU kartları daha sonra farklı alanlarda kullanılmaya başlandı.

He ve ark. (2010), CUDA'yı kullanarak tasarladıkları sırlama işlemini de içeren birleştirme(join) algoritması ile CPU'ya oranla 2-7 kat arasında başarım elde etmişlerdir. Yine, Bakkum ve Skadron (2010), sanal makine altyapısı içerisinde SQLite'ın GPU üzerinde yürütülecek şekilde tüm SQL sorgularının çok farklı bir yaklaşım GPU üzerinde işlenmesi ile gerçekleşen basit sorgularda SQLite'a oranla 20x daha hızlı bir şekilde işlem yapabilmişlerdir.

Wu (2015), GPU belleğinin sınırlı olan kapasite dikkate alındığında veri üzerinde uygulanan algoritmalar büyük veri kümeleri için doğrudan kullanılamaz. CUDA birleşik sanal adresleme ile GPU'nun doğrudan CPU belleğinde yer alan veriye ulaşılabilmesi ile verinin GPU'ya kopyalanması bir zorunluluk değildir. Bu durumda da algoritmanın sunacağı performans PCIe bant genişliği sınırına bağlı olarak artacak veya azalacaktır.

Cai (2015) GPU temelli benzetim sisteminde, hesaplama süresinde ve hesaplama maliyetinde kayda değer bir düşüş ile birlikte benzetimde daha fazla eleman örgüsünü mümkün kılarken, bu elemanların geometrik olarak daha iyi gösterilmesini ve daha kesin sonuçlar elde edilmesini sağladığından bahsetmiştir. GPU'lara uygulanmak üzere geliştirilen hassas kütüphane ile bilimsel uygulamalar için GPU temelli sorgu aracı ile önemli bir performans artışı sağlanması amaçlamıştır (Lu ve ark., 2010).

Nezarat (2005) tarafından yapılan çalışmada hesaplama süresi bakımından en popüler ve yüksek performanslı yöntemlerden birisi grafik işlemciler dizisi ile çok

yüksek hızlı hesaplama gücü imkânı sunar. Aynı zamanda, düşük hesaplama güçlü ve çok çekirdekli GPU işlemci dizisi yüksek hızlı hesaplama gücü ve bellek okuma/yazma hızı bakımından süper bilgisayarlar veya kümeler arasındaki şebeke bağlantı hızından daha ileri düzeydedir.

Abecassis (2015) yaptığı çalışmada, CUDA mimarisi ile hesaplama süresinde önemli ölçüde artışı sağlanabileceği fakat görev parçacıkları doğru bir şekilde ayarlanmaz ise güçlü paralel CUDA mimarisi kullanılmamalıdır. Çünkü bu durumda uygulama CPU'da daha iyi sonuçlar verecektir. Büyük ölçekli veya küçük ölçekli işlerde paralelleştirme stratejisinin performans açısından istenen sonucu verebilmesi için uyarlama işleminin CUDA işleyişine uygun bir şekilde yapılması gerektiğini belirtmiştir.

Rauhe (2015), ilişkisel sorguların GPU üzerinde çoklu-düzey paralelleştirme ile yürütülmesi için geliştirdikleri kütüphanede metin operatörlerinin desteğinin olmamasından dolayı sadece bazı sorgu gerçekleştirebildiler. Performansın arttırılabilmesi için farklı yöntemler denediler fakat yük dengelemesi açısından başarılı bir optimizasyonun sağlanamamasından dolayı performans açısından olumlu bir sonuç elde edemediler.

He ve ark. (2009), yaptıkları çalışmada GPU ve CPU arasındaki veri alışverişi sınırlı bir bant genişliği ile geçekleşir. CPU, GPU çekirdekleri için dinamik bellek tahsisine izin vermemesinden dolayı dikkatli bir tasarım gerektirmektedir. GPU tabanlı algoritma CPU temelli emsalleri ile kıyaslandığında 2-27x arası performans artışı gerçekleşmiştir.

Pandey ve Tokekar (2014), iş dünyasının daha iyi hizmet verebilmesi için büyük verilerin derlenmesi ve makul çıkarımlarda bulunması organizasyonların karar alma sürecinde olumlu bir değişim ve büyük bir avantaj sağlamaktadır, fakat oluşan devasa verilerin derlenmesi çeşitli zorlukları da beraberinde getirmektedir. Büyük verilerin işlenmesinde MapReduce paradigmasının esnek, ölçeklenebilir ve verimli sonuç vermesinin diğer geleneksel araçlara göre ne kadar önemli rol oynadığını açıklamaktadır.

Breß ve ark. (2012), geliştirdikleri HyPE ile kendi kendini ayarlayabilen sorgu düzenleyici ile sorgunun GPU veya CPU üzerinde yürütülmesine karar vererek melez çözümlerin performans açısından daha iyi bir çözüm olacağını savunmuşlardır. GPU ile yürütülen gizli-önemli verilerin güvenlik sorunları olduğunu fark etmişlerdir.

Malakar ve Vydyanathan (2013), büyük veri analizi özellikle metin analizi alanlarında günümüz endüstrisinde sıkça kullanılan Hadoop MapReduce ile GPU'ların fiyat-performans oranlarının paralel hesaplama gücü dikkate alındığında oldukça cazip olmasından dolayı, GPU'nun paralel işlem yeteneğini Hadoop'un metin analiz özelliği ile resimlerin işlenerek yüz tanıma algoritmasında performansın %25 iyileştirilebildiğini göstermiştir.

Ilic ve ark. (2011), yılında yaptıkları çalışmada zamanlama kütüphanesi ile yapılmak istenen işlemin CPU veya GPU üzerinde yürütülmesine bağlı olarak iki işleme merkezinin birlikte kullanılmasının performans artışına önemli katkıları olacaktır.

Yuan ve ark. (2013), CPU ve GPU'da gerçekleştirilen sorguların detaylı analizlerinin yer aldığı bu çalışmada, GPU'nun sadece bellekte sabitlenebilen belirli sorgularda kullanılabilmesinden dolayı veri depolama sistemlerinde uyum sağlamasında yaşan problemlere değinmişlerdir.

Heimel ve Markl (2012), sorguların optimizasyonu minimum veri aktarımını ile birlikte ağır bir iş yükünün tamamlanmasını da gerektirir. GPU destekli tahminlerin PostgreSQL optimizasyonuna entegre edilmesiyle GPU'nun ilişkisel veri tabanı sisteminde seçicilik tahminlerinin kalitesinin arttırılabilmesi için verimli bir şekilde kullanılabileceğini belirtmişlerdir.

Sharma ve ark. (2018), büyük hacimli verilerin işlenmesi, depolanması ve yönetimi klasik veri yönetim araçları ile çok başarılı sonuçlar vermemesinden dolayı ölçeklenebilir ve dağıtılmış sistem ile verilerin paralel bir şekilde işlenmesi ile yönetim daha makul sonuçlar verebilecektir. SPARQL sorgularının Spark SQL ile yürütülmesi ile sıkıştırılış veriler üzerinde gerçekleştirilen çalışmada yanıt süresinin azaldığını göstermişlerdir.

Zaharia ve ark. (2012), veri kümelerini bellek içinde tutarak verinin verimli kullanımını amaçlayan Spark, SQL sorguları, makine öğrenmesi ve grafik ve akış analitiği gibi ayrı ayrı bileşenlerden oluşmaktadır. Veri güdümlemesine ilişkin bütün uygulamaları çalıştırabilen Spark, öncekilerden farklı olarak yinelemeli veri akışı modellerini de destekleyen genel amaçlı bir veri analitiği çerçevesidir.

Gonzales (2014), birbirileriyle anlamlandırılabilir ilişkiler ile bağlantılı olan veriler devasa boyutlara ulaşabilmektedir. Kullanıcılara sağladığı imkânlar açısından büyük bir potansiyel taşımasına karşın uygulanmalarında birçok zorluk ile karşılaşılmaktadır. BIPFRAME kütüphanesi ile kaynakların web ile ilişkilendirilerek tüm kullanıcıların erişebilmesini sağlamıştır.

Li ve ark. (2015), büyük veri kümeleri üzerinde hesaplama vb. işlemler gerektiren analiz algoritmalarının GPU hızlandırıcılı heterojen yapı Spark ile bütünleştirilerek GPU'nun devasa hesaplama kapasitesini, CPU ve belleğin optimize edilerek kullanılmasıyla gerçekleştirmişlerdir. Geliştirdikleri HeteroSpark platformu ile farklı farklı makine öğrenmesi uygulamalarında 18x'a kadar performans artışı gözlemlemişlerdir. Tek GPU ile yapılan denemelerde bellek boyutunun yetersiz olması nedeniyle büyük veri kümeleri ile mümkün olmazken, çoklu GPU ile yapılan denemelerde ise veri bölümlenmesindeki karmaşıklığa değinmişlerdir.

Manzi ve Tompkins (2016), Spark ana işlemlerinin GPU'nun kullanılarak gerçekleştirilmesini sağladıkları çalışmalarında, k-kümeleme algoritmasının uygulanmasında 17x'a kadar hızlandırma elde etmişlerdir. Bu başarımda verinin GPU'da işlenebilecek hale getirilmesi ayrı bir çaba gerektirdiğinden performans açısından bir maliyet ortaya çıkmaktadır.

Shyam ve ark. (2015), büyük veri uygulamaları için küme bilgi işleme platformu olarak ortaya çıkmıştır. Akıllı şebeke uygulamaları çok çeşitli büyük veri işleme yöntemleri gerektirdiğinden basit MapReduce kalıplarıyla yapılan işlemler yetersiz kalmaktadır. Akıllı şebeke veri platformu hızlı veya yavaş her türlü veri değişimini analiz edebilecek yeterlilikte olmalıdır. Spark, gerçek zamanlı ve tekrar eden veri işleme ihtiyaçlarına sorunsuz bir şekilde yerine getirebilmektedir.

Ohno ve ark. (2016), Spark yapsının CUDA çekirdekleri kullanılarak gerçekleştirilen yoğun hesaplama gerektiren işlemlerde, RDD'lerin GPU üzerinde yrütülmesi için dizilere dönüştürülmesi, ve GPU bellek kapasitesinin en verimli şekilde kullanılmasını sağladılar. Bu çalışma neticesinde indirgeme ve dönüştürme farklı farklı denemelere karşı yapılan değerlendirmelerde GPU performans bakımından normal yazılıma göre 21.4x daha hızlı sonuç vermiştir.

Wang ve ark. (2014), yükse çıktılı işlemlerde GPU'nun eş zamanlı sorgularda etkili kullanımını ve paylaşımın yol açtığı kaynakları kullanma isteklerinin yazılım desteğiyle çözmeye çalışmışlardır. GPU sorgu zamanlama ve bellek değiştirme politikasından yola çıkarak sistem çıktısında %55 kadar gelişme kaydettiklerini belirtmişlerdir.

Plase ve ark., (2017), Avro, Parquet ve diğer metin formatlarının veri sorgularındaki performans farklılıklarının irdelendiği çalışmalarında, Avro ve Parquet'in binary veri formatında verileri tutması ve sıkıştırma avantajlarından yararlanmasından dolayı depolama açısından daha az yer kapladıklarını gözlemlemişlerdir. Bir diğer önemli nokta olan veri sorgulama işlemlerinin ise sütun tabanlı veri format olan Parquet'in Avro ve diğer metin formatlarına oranla daha hızlı gerçekleştirdiğini belirtmişlerdir.

Haynes ve ark. (2015), Apache Spark Parquet'in sütun tabanlı veri saklama için kullanılan veri tabanını yeni nesil Terra Populus tablo haline getiren uygulamalarında kullandılar. Veri türlerine göre sıkıştırma oranları açısından iyi sonuçlar verdiğini belirtmişlerdir.

Awan ve ark. (2016), Hadoop sistemlerinde SQL sorgularının donanım ve yazılım bakımından kıyaslandığı çalışmada ORC, TXT ve Parquet veri formatlarımdaki veri dosyaları ile ilgili performans farklılıklarına değinmişlerdir. Bu çalışmada Impala ile TXT dosyaları üzerinde yapılan sorgularda Parquet'e oranla yaklaşık olarak 9x daha hızlı sonuç alınmıştır.

Abadi ve ark., (2008), sütun tabanlı ve satır tabanlı veri tabanlarının performans ve avantajları açısından incelendiği bu çalışmada, performans farklıklarının uygulayıcı

düzeyindeki farklılıkları analiz edilmiştir. Satır tabanlı veri tabanları için de sütun tabanlı veri tabanlarındaki performans ve avantajları yakalanması mümkündür fakat depolama aşamasında ve sorgu uygulama düzeyinde bazı değişikliklerin yapılması gerektiğini belirtmişlerdir.

3. MATERYAL ve YÖNTEM

Büyük verilerin Apache Spark ve GPU üzerinde gerçekleşen sorguların analizlerinin gerçekleştirildiği bu çalışmada, sonuçları bakımından belirleyici olacak olan bileşenler kullanılan materyal ve izlenen yöntem bölümlerinde detaylarıyla yer almaktadır.

3.1. Materyal

Büyük verilerin GPU üzerinde işlenmesi ile performansın ölçüleceği bu çalışma için gereken materyaller donanım, yazılım ve veri olmak üzere 3 (üç) ana kategoride sınıflandırılabilir.

3.1.1. Donanım

Bilgisayar sisteminin ana kart üzerinde bulunun bütün bileşenler donanım olarak adlandırılmaktadır (Şekil 3.1.). Günlük hayatta kullanılan bilgisayarlardan farklı olarak bu çalışmada genel amaçlı GPU kullanılacaktır. Normalde CPU üzerinde yürütülen sorgular yerine, giriş bölümünde de detaylarına değindiğimiz GPU'nun performansa olan etkileri çalışmamızda büyük önem taşımaktadır.

Şekil 3.1. Çoklu GPU iş istasyonu

Farklı özellik ve donanımlarda üretilen GPU'lar kişisel bilgisayarlarda ekran kartı olarak bulunabilirken GPGPU'lar daha çok iş istasyonları ve sunucularda hesaplama işlemlerinde, GPU'ya ihtiyaç duyulan uygulamalar veya bilimsel araştırmalarda kullanılmaktadır. Giriş bölümünde de bahsedildiği gibi NVIDIA tarafından farklı özelliklerde GPU kartları üretilmiştir. Bu çalışmada ise Çizelge 3.1.'de özellikleri belirtilen Tesla T4 GPGPU hızlandırıcı kullanılacaktır.

Çizelge 3.1. Tesla T4 GPGPU özellikleri

Özellik	
GPU mimarisi	NVIDIA Turing
NVIDA Tensor Çekirdeği Sayısı	320
NVIDIA Cuda Çekirdeği Sayısı	2560
Single Precision	8.1 TFLOPS
Mixed Precision	(FP16/FP32) 65 TFLOPS
INT8	130 TOPS
INT4	260 TOPS
GPU Memory (GPU Belleği)	16 GB GDDR6 300 GB/sec
ECC	Evet
Interconnect Bandwidth	32 GB/sec
System Iterface (Sistem Arayüzü)	x16 PCIe Gen3
Form Factor (Anakart Boyutu)	Low-Profile PCIe
Thermal Solution(Termal Çözüm)	Pasif
Compute APIs	CUDA, NVIDIA TensorRT™, ONNX

Tesla T4'ün tercih edilmesinin nedeni ise hem yüksek işlem gücü hem de GPU SQL sorgularının gerçekleştirilebilmesi için kullandığımız BlazingSQL kütüphanesinin bu GPU aygıtı ile uyumlu çalışmasındır.

İş istasyonunda yer alan diğer donanımlar ise 12GB DDR4 RAM ve Çizelge 3.2'de özellikleri belirtilen Intel Xeon CPU'dur.

Çizelge 3.2. CPU özellikleri

Özellik	
Cpu MHz	2200
L1d bellek	32K
L1i bellek	32K
L2 bellek	256K
L3 bellek	56320 K
Adres Boyutu	46 bit fiziksel, 48 bit sanal

Çizelge 3.3. **(devam)**

İşlemci Ailesi	6
Çekirdek Sayısı	1

3.1.2. Yazılım

İş istasyonumuzda Linux tabanlı Ubuntu işletim sistemi bulunmaktadır. Sorgulama işlemlerinin gerçekleştirilebilmesi için kullanılan Python uygulamamızın ihtiyaç duyduğu diğer bileşenler Rapids tarafından açık kaynak olarak sunulan cuDF ve BlazingSQL kütüphaneleri, Apache Yazılım Kuruluşu tarafından açık kaynak olarak sunulan Apache Spark kütüphanesi ve NVIDIA'nın geliştiricilere yüksek performanslı GPU uygulamaları geliştirme olanağı sunan CUDA Toolkit kullanılacaktır. Çizelge 3.3.'te ise kullanılan işletim sistemi ve kütüphanelere ilişkin sürüm bilgileri yer almaktadır.

Çizelge 3.4. Kullanılan yazılım kütüphaneleri sürümleri

Kütüphane	**Sürüm**
Ubuntu İşletim Sistemi	18.04.2 LTS
Python	3.6.8
Apache Spark	2.4.3
cuDF	0.8
CUDA Toolkit	9.2

3.1.3. Veri

Büyük verilerin analizine ilişkin sorgu performanslarının analiz edileceği bu çalışma da ihtiyaç duyulan diğer bileşen ise verilerdir. Bu kapsamda Freebase API tarafından CC-BY lisansı altında sunulan 2013 Mart ayı boyunca gerçekleşen silme işlemi aktivite kayıtları(log) kullanılacaktır. Bu aktivite kayıtlarında silme işlemlerin kim tarafından ne zaman gerçekleştirdiğine ilişkin yaklaşık olarak 62.000.000 satır ve ham haliyle yaklaşık olarak 8GB boyutunda olan ve Çizelge 3.4.'te alan adları belirtilen veriler bulunmaktadır.

Çizelge 3.5. Veri tabanında yer alan alanlar

	Alan Adları
1.	creation_timestamp (Unix epoch time in milliseconds)
2.	creator
3.	deletion_timestamp (Unix epoch time in milliseconds)
4.	deletor
5.	subject
6.	predicate
7.	object
8.	language_code

3.2. Yöntem

Uygulamamızda kullanılacak olan CUDA Toolkit, cuDF, BlazingSQL ve Apache Spark kütüphanelerinin kurulumu gerçekleştirildikten sonra verilerin hazırlanması ve analizlere dair çıktıları verecek Python uygulamalarının işleyişine ilişkin detaylar bu bölümde yer almaktadır. Sırasıyla verilerin hazırlanması, sorgu türlerinin belirlenmesi, Apache Spark sorgularının gerçekleştirilmesi ve GPU sorgularının gerçekleştirilmesi neticesinde uygulama süreci tamamlanacaktır.

3.2.1. Analiz edilecek verilerin hazırlanması

Büyük verilerde sorgulardaki performansın en iyi sonucu verebilmesi için verilerin GPU'nun işleyişine de uygun olan sütun yönelimli verilerin kullanılması gerekmektedir.

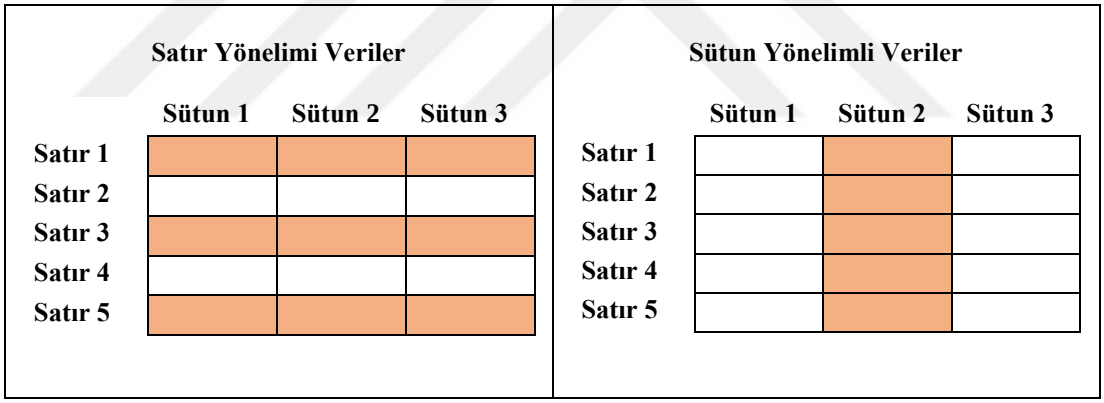
Problem Tanımı bölümünde nedenlerine değinildiğimiz veriler 1(bir) aylık aktivite kayıtları olacaktır. CSV formatındaki aktivite kayıtlarımız hem Apache Spark hem de GPU uygulamamızda kullanılabilecek Parquet formatına nasıl dönüştürüldüğü Çizelge 3.5.'te belirtilmektedir. Apache Spark DataFrame fonksiyonları veri format dönüştürme işlemlerini kolay bir şekilde gerçekleştirmektedir. Dönüştürme sırasında CSV formatına dönüştürülmüş verilerde hem sütun sınırlayıcı olarak virgül karakteri kullanılması hem de "subject" alanı içerisinde de kullanılması nedeniyle sütun sayısının bazı satırlarda artması hataya neden olduğundan bu satırlar ayrı bir Python uygulaması ile düzeltilerek dönüştürme işlemi tamamlanmıştır. Parquet veri formatı özellikle belirli bir sütun için işlem yapılması gerektiğinde performans yönünden çok başarılı sonuçlar vermektedir.

Çizelge 3.6. CSV dosyasının Parquet dosyasına dönüştürülmesi

```
1. …
2. df = pd.read_csv('/deletions.csv')
3. df.to_parquet(' /deletions.parquet')
4. …
```

Başlangıçta 8GB büyüklükte olan ham CSV dosyamız Parquet formatına dönüştürüldükten sonra yaklaşık olarak 2GB boyutunda yeni bir veri dosyası oluşmuştur. Veri boyutunun azalması Parquet'in verileri sütun yönelimli olarak binary formatında tutması ve sıkıştırmasından kaynaklanmaktadır.

Sütun yönelimli veri tabanları ile satır yönelimli veri tabanları arasındaki tek farklılık, satır yönelimli verilerin bulunduğu bir satır çağırıldığında satırda ihtiyaç duyulmayan verilerin de işleme dâhil edildikten sonra istenmeyen bölümün filtrelenmesi ile hedef veriye ulaşılabilirken, sütun yönelimli verilerde ise sadece ilgili sütunun istenen satırı çağırılmaktadır (Şekil 3.2.).



Şekil 3.2. Satır ve Sütun yönelimli veri tabanları

3.2.2. Sorgu türlerinin belirlenmesi

Veri tabanlarında yer alan tablolardan istenen verilerin elde edilmesini sağlayan sorgu türleri bu bölümde yer almaktadır. Uygulamalar da kullanılacak sorgular genel itibariyle birbirine benzeler, farklılıklar ise işlevden çok fonksiyon isimlerinden kaynaklanmaktadır. Bu çalışmada Çizelge 3.6.'da yer alan 7 farklı sorgu gerçekleştirilecektir. Bu sorguların tercih edilmesinin nedeni birçok uygulamada yaygın olarak kullanılan temel sorgular olmasıdır.

Çizelge 3.7. Sorgu türleri

1.	Tüm alanların yer aldığı ilk 10 kaydın seçimi
2.	Herhangi bir alana göre sıralama
3.	Belirli alanların seçildiği ilk 10 kayıt
4	Toplam kayıt sayısı
5.	Herhangi bir alandaki farklı kayıt sayısı
6.	Belirli bir alana göre gruplandırma ve ilgili gruptaki kayıt sayısı
7.	Belirli bir koşula göre seçim yapılması

Çizelge 3.6.'da belirtilen sorgu türlerinin Python uygulamamızdaki SQL, Spark DataFrame ve cuDF DataFrame sorguları Çizelge 3.7.'de ayrıntılı olarak yer almaktadır.

Çizelge 3.8. Sorgu cümleleri

1.	SQL	SELECT * FROM Deletions LIMIT 10
	cuDF DataFrame	cuDF.head(10)
	Spark DataFrame	sparkDF.limit(10)
2.	SQL	SELECT * FROM Deletions ORDER BY creation_timestamp
	cuDF DataFrame	cuDF["creator"].sort_values(['creator'])
	Spark DataFrame	sparkDF.orderBy("creation_timestamp")
3.	SQL	SELECT creator, deletor FROM Deletions LIMIT 10
	cuDF DataFrame	cuDF[["creator","deletor"]].head(10)
	Spark DataFrame	sparkDF.select("creator","deletor").limit(10)
4	SQL	SELECT COUNT(*) FROM Deletions
	cuDF DataFrame	cuDF["creator"].count()
	Spark DataFrame	sparkDF.count()
5.	SQL	SELECT COUNT(DISTINCT creator) FROM Deletions
	cuDF DataFrame	cuDF["creator"].unique().count()
	Spark DataFrame	sparkDF.select("creator").distinct().count()
6.	SQL	SELECT COUNT(*) as NumOfEvents, creator FROM Deletions GROUP BY creator
	cuDF DataFrame	cuDF.groupby("creator").agg({"creator":"count"})
	Spark DataFrame	sparkDF.groupBy("creator").count().select("creator","count")
7.	SQL	SELECT * FROM Deletions WHERE deletor = '/user/funderhill'
	cuDF DataFrame	cuDF. query('deletor == '/user/funderhill')
	Spark DataFrame	sparkDF.where("deletor == '/user/funderhill'")

3.2.3. Apache Spark sorgularının gerçekleştirilmesi

Bu bölümde Apache Spark kullanılarak hem SQL hem de Python/R dillerinde desteklenen DataFrame fonksiyonları kullanılarak sorgular gerçekleştirilecektir (Şekil 3.3. ve Şekil 3.4.). Aynı sorgular farklı zamanlarda belirli sayılarda tekrarlandıktan sonra her işlemin gerçekleşme süreleri ayrı ayrı kaydedilmiştir.

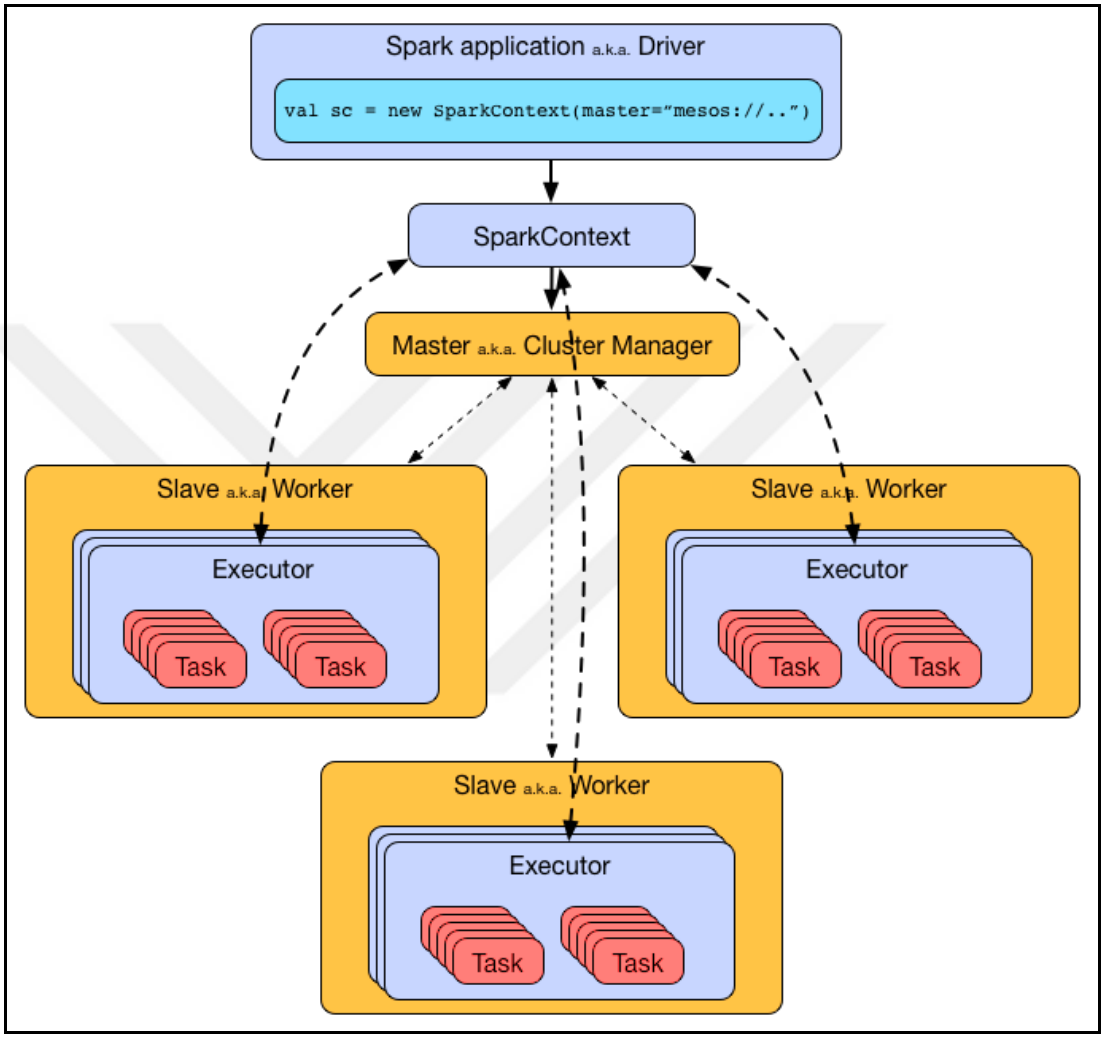
```
%%time
result = spark.sql("SELECT * FROM Deletions LIMIT 10")

CPU times: user 844 µs, sys: 1.18 ms, total: 2.02 ms
Wall time: 21.7 ms

%%time
result = sparkDF.limit(10)

CPU times: user 770 µs, sys: 967 µs, total: 1.74 ms
Wall time: 5.75 ms
```

Şekil 3.3. Spark SQL ve Spark DataFrame Python 1. Sorgu

```
%%time
result = spark.sql("SELECT creator, deletor FROM Deletions LIMIT 10")

CPU times: user 2.3 ms, sys: 59 µs, total: 2.36 ms
Wall time: 18.4 ms

%%time
result = sparkDF.select("creator","deletor").limit(10)

CPU times: user 2.76 ms, sys: 748 µs, total: 3.51 ms
Wall time: 20.2 ms

print(result.show(5))

+--------------------+--------------------+
|             creator|             deletor|
+--------------------+--------------------+
|/user/mwcl_musicb...|/user/google_gard...|
|/user/mwcl_wikipe...|/user/mwcl_wikipe...|
|/user/mwcl_musicb...| /user/turtlewax_bot|
|   /user/mwcl_images| /user/gardening_bot|
|/user/mwcl_musicb...|/user/mbz_pipelin...|
```

Şekil 3.4. Spark SQL ve Spark DataFrame Python 3. Sorgu

Spark ile işlem yapılacak veri tabanı Driver aracılığıyla seçilir daha sonra Spark sorguları SparkContext yardımıyla anlamlandırılarak Cluster Manager yardımıyla ilgili hesaplamalar yürütücülerin yer aldığı JVM Node'ları yani Worker'lar üzerinde görevler yerine getirilerek işlem tamamlanır (Şekil 3.5.).



Şekil 3.5. Apache Spark sorgu iş akışı

3.2.4. GPU sorgularının gerçekleştirilmesi

SQL sorguları Apache Spark için kullanılan sorgulara benzeyen GPU sorguları BlazingSQL ile SQL, cuDF ile DataFrame fonksiyonları kullanılarak gerçekleştirilmiştir (Şekil 3.6. ve Şekil 3.7.). cuDF DataFrame fonksiyonları işlev olarak aynı ama bazı fonksiyon isimlerinde farklılık gösterebilmektedir. Yine Spark sorgularındaki gibi farklı zamanlarda ve aynı sayıda tekrarlanarak gerçekleştirilmiştir.

```
%%time
result = bc.sql('SELECT * FROM main.deletions LIMIT 10')

CPU times: user 3.69 ms, sys: 0 ns, total: 3.69 ms
Wall time: 14.2 ms

%%time
result_get=result.get()
print(result_get.columns)

%%time
result =cuDF.head(10)

CPU times: user 15.5 ms, sys: 3.18 ms, total: 18.6 ms
Wall time: 20.4 ms

print(result)

    creation_timestamp                 creator  deletion_timestamp
0        1354923961007   /user/mwcl_musicbrainz       1356010979000         /user/googl
1        1352854086000  /user/mwcl_wikipedia_en       1352855856000       /user/mwcl_wi
2        1355171076000   /user/mwcl_musicbrainz       1364258198000          /user/tur
```

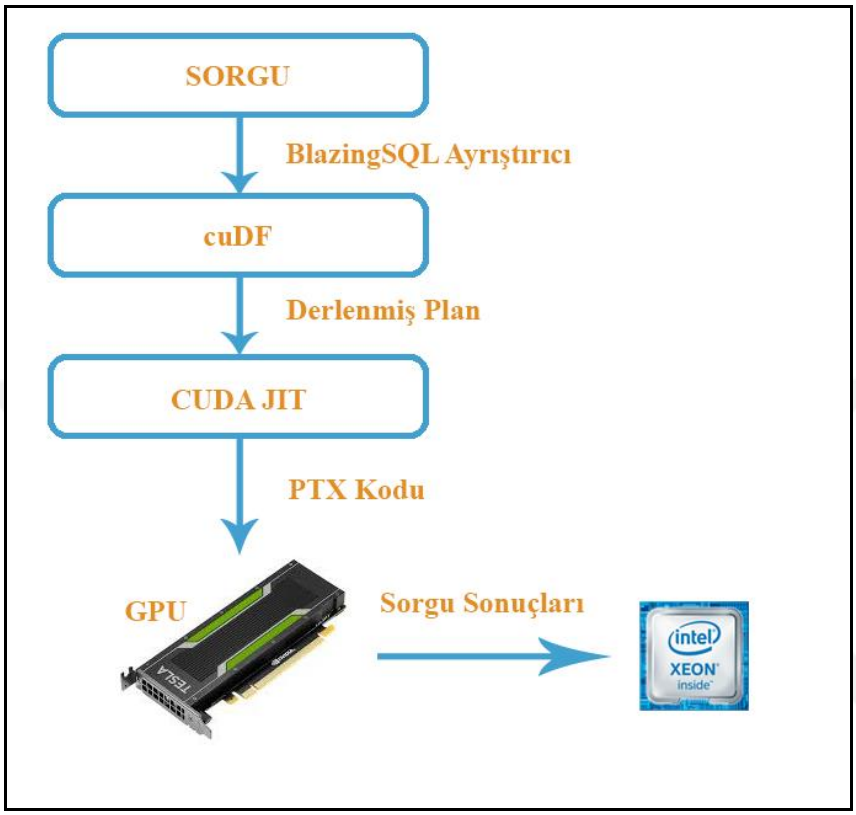
Şekil 3.6. SQL ve cuDF ile DataFrame Python 1. Sorgu

```
%%time
result = bc.sql("SELECT COUNT(*) FROM main.Deletions")

CPU times: user 3.56 ms, sys: 280 µs, total: 3.84 ms
Wall time: 16.5 ms

%%time
result = cuDF["creator"].count()

CPU times: user 465 µs, sys: 0 ns, total: 465 µs
Wall time: 472 µs

print(result)

62711522
```

Şekil 3.7. SQL ve cuDF ile DataFrame Python 4. Sorgu

GPU üzerindeki iş akışı BlazingSQL ile SQL sorgusu cuDF üzerinde derlenebilecek şekilde en uygun hale getirildikten sonra Python fonksiyonları PTX koduna dönüştürülmektedir. Daha sonra CUDA JIT ile PTX koduna dönüştürülen

istek son olarak Python Numba ile CUDA sürücü API'sini kullanarak CUDA aygıtı üzerinde sorgu işleminin gerçekleştirilmesini sağlar ve sonuçlar CPU'ya aktarılır (Şekil 3.8.).



Şekil 3.8. GPU sorgu iş akışı

PTX, CUDA programlama ortamlarında kullanılan pseudo-assembly dilidir. Nvcc ile C++ benzeri yazılım dillerinde yazılan CUDA kodlarını PTX formatına dönüştürürken, CUDA Jit ile Python yazılım dillerinde benzer işlevi yerine getirmektedir.

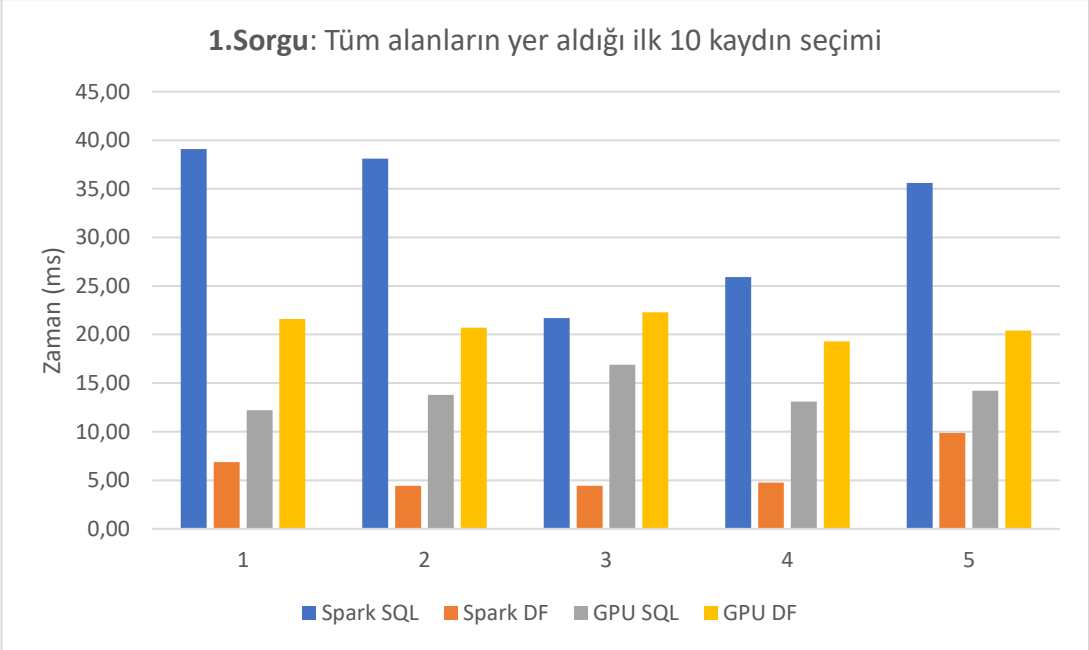
4. ARAŞTIRMA BULGULARI ve TARTIŞMA

Bu bölümde sorgulara ilişkin performans karşılaştırmaları yer almaktadır. Yöntem bölümünde yer alan Çizelge 3.4.'deki sorgu türleri Spark SQL, Spark DataFrame, GPU SQL ve GPU DataFrame ile ayrı ayrı gerçekleştirilmiştir. Bulgular bölümünde bu verilere ilişkin detaylar, tartışma bölümünde ise elde edilen sonuçlar ile ilgili analizler yer almaktadır.

4.1. Araştırma Bulguları

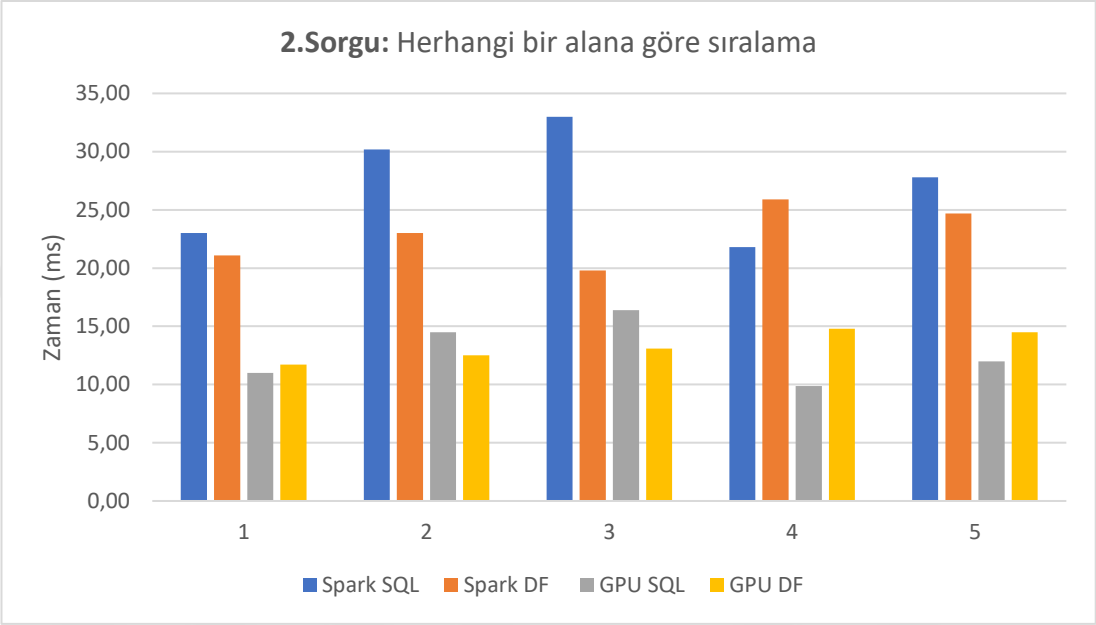
Performans farkının incelendiği bu bölümde, sorgu işlemlerinin gerçekleşme süreleri dikkate alınmıştır.

Herhangi bir koşul veya birleştirme, toplama vb. işlemin olmadığı sorgular yoğun işlem gerektiren sorgular değildir. Şekil 4.1.'de yer alan grafikte Spark DataFrame ile gerçekleştirilen sorgu daha kısa sürede gerçekleşmiştir ve GPU sorgularına oranla yaklaşık olarak 3x, Spark SQL'e oranla 6x daha hızlı yanıt vermiştir. GPU SQL'in performansı Spark SQL ile kıyaslandığında sonucun ortalama olarak 2.2x daha hızlı sonuçlandırdığı görülmüştür.



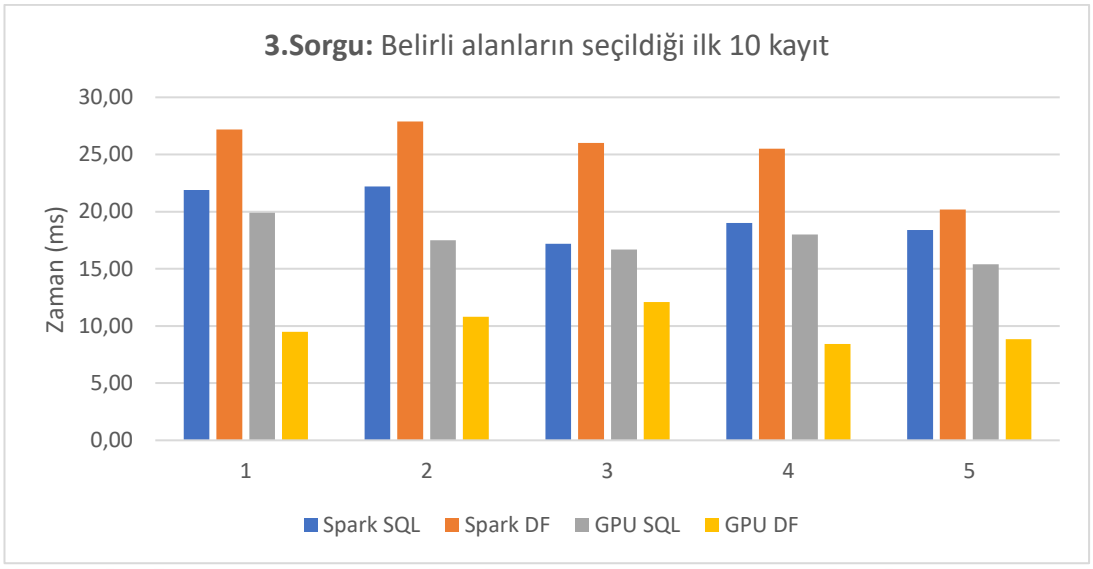
Şekil 4.1. Sorgu 1 performans karşılaştırma grafiği

Sıralama işlemi ise işlem gerektiren bir eylem olmasından dolayı GPU'nun hesaplama kapasitesinin yansıtılmasında daha etkili olabilecektir. Şekil 4.2.'de yer alan grafikte de görüldüğü gibi GPU üzerinde yürütülen sorgular ise ortalama değerler dikkate alındığında GPU'nun sorguyu 2x daha hızlı şekilde yerine getirdiği görülmüştür.



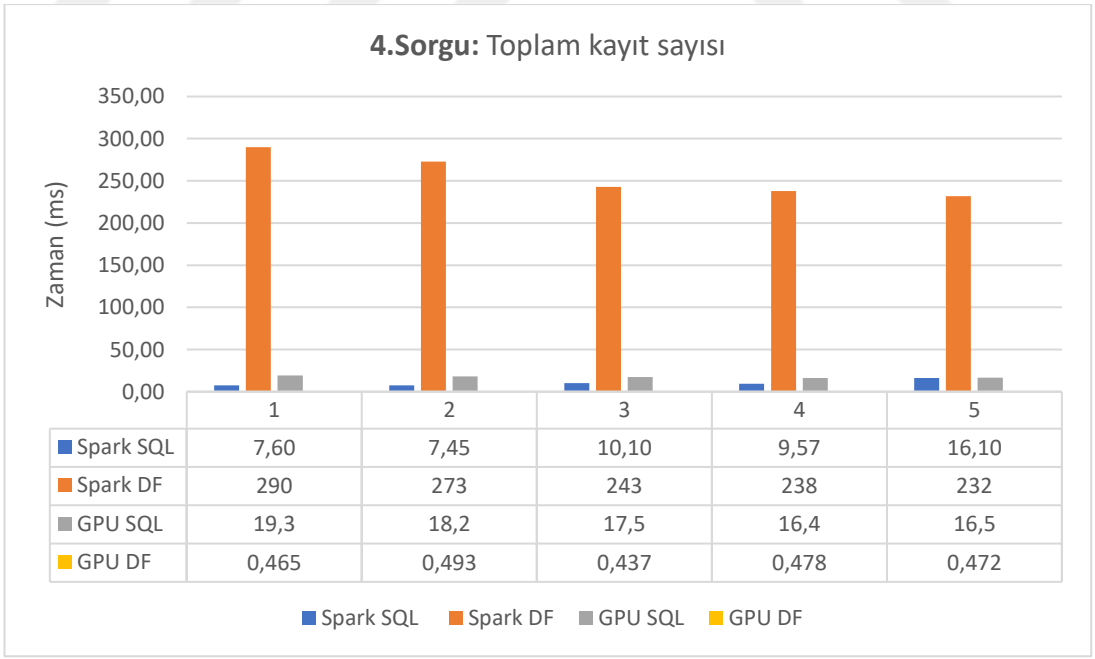
Şekil 4.2. Sorgu 2 performans karşılaştırma grafiği

Veri dosyamızdaki alanlardan bazılarının seçilerek sorgulandığı bu sorguda ise SQL sorguları arasında anlamlı bir fark görülmemekte fakat DataFrame performansları arasında 2x-3x arasında değişen performans farkı ile gözlenmiştir(Şekil 4.3.).



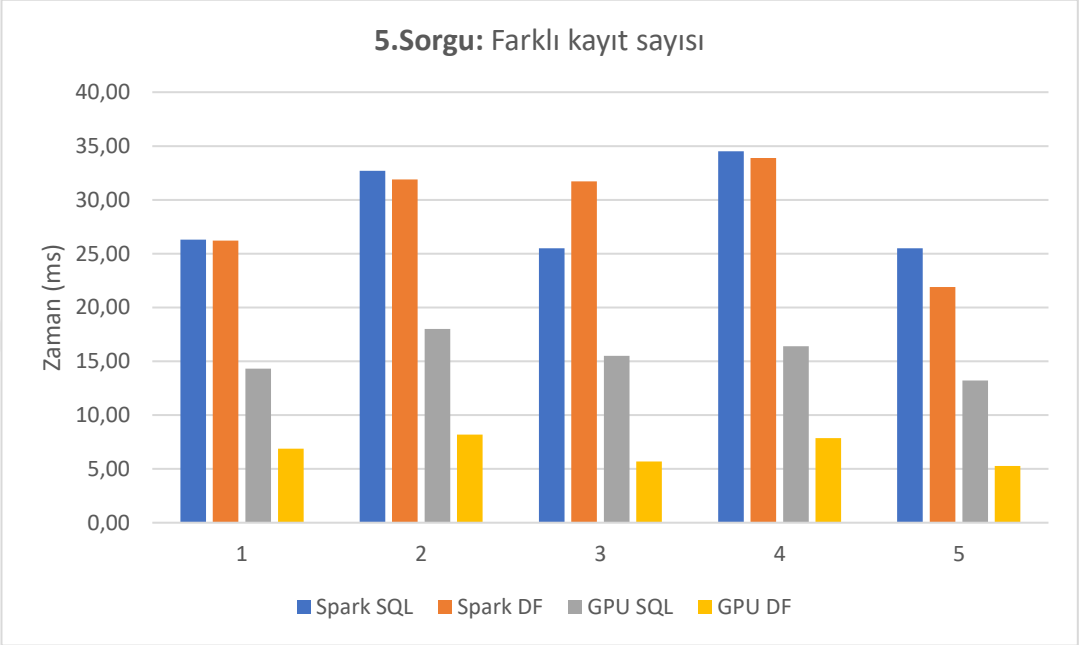
Şekil 4.3. Sorgu 3 performans karşılaştırma grafiği

GPU'nun performansı bakımından en büyük farklılığın gözlemlendiği sorgu, Şekil 4.4.'de yer alan ve toplam kayıt sayısının görüntülenmek istediği bu sorguda ortaya çıkmaktadır. GPU DataFrame Spark SQL'e göre ortalama 25x daha hızlı, Spark DataFrame ile arasında 500x gibi önemli bir performans farkına rastlanmıştır.



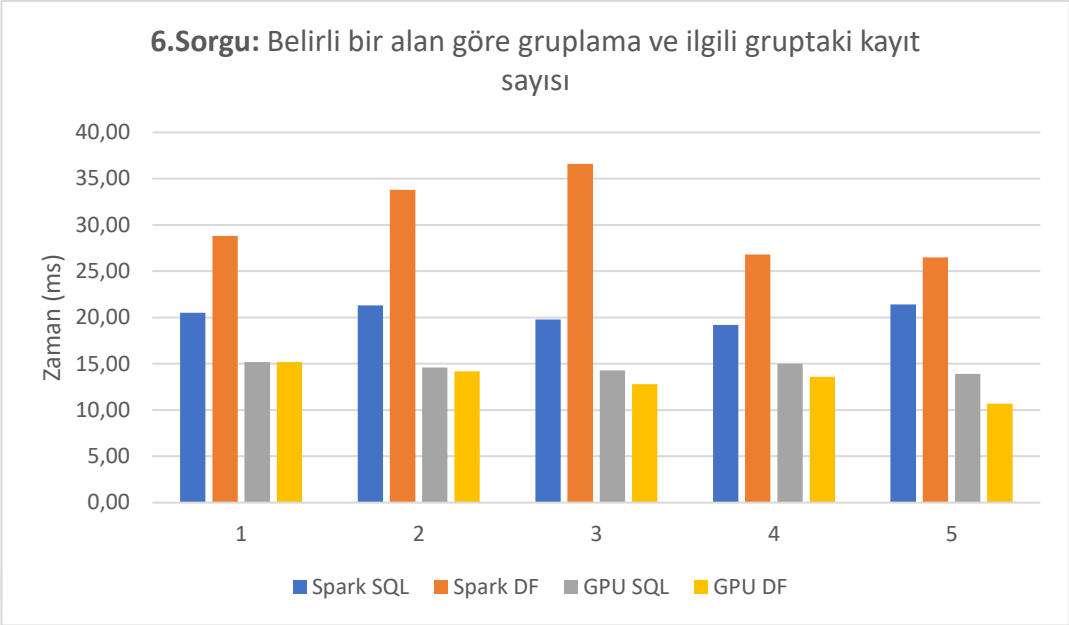
Şekil 4.4. Sorgu 4 performans karşılaştırma grafiği

Bir diğer önemli sorgumuz ise belirli bir alandaki farklı kayıtların seçilmek istendiği 5. Sorgu' dur. Şekil 4.5.'te yer alan verilere göre SQL performansları bakımından GPU SQL sorgusu ortalama 2x daha hızlı gerçekleşmiştir. DataFrame'ler ile gerçekleştirilen sorgularda ise GPU 4.3x daha kısa sürede sonuç vermiştir.



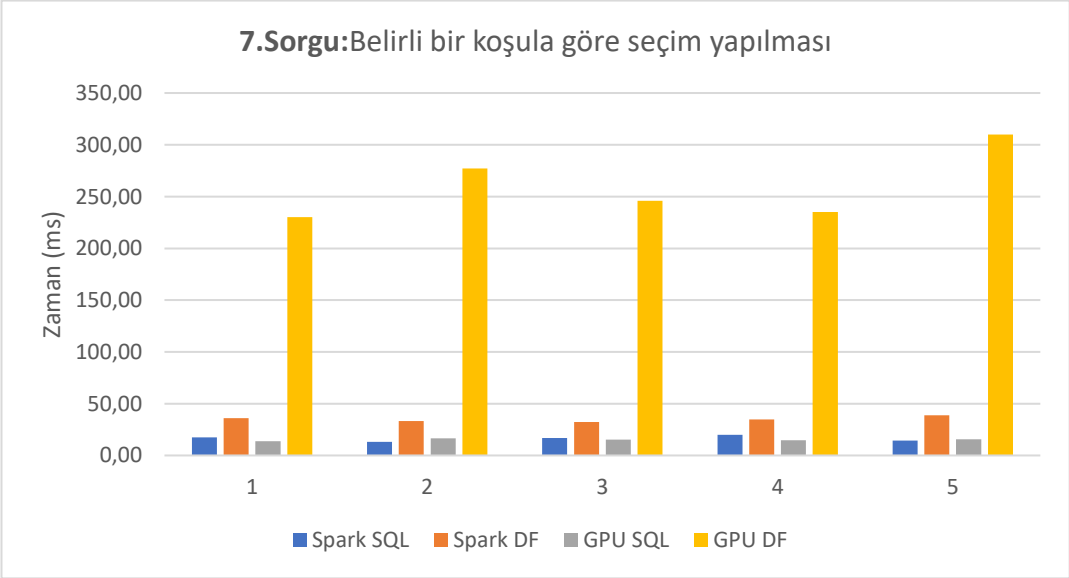
Şekil 4.5. Sorgu 5 performans karşılaştırma grafiği

Uygulamalarda yaygın olarak kullanılan sorgu türlerinden biride gruplandırmaya bağlı olarak grup içindeki öğelerin genelde aritmetik işlemlerin gerçekleştirildiği sorgulardır. Şekil 4.6.'da ise belirli bir grupta yer alan öğe sayısı sorgulanmıştır. GPU DataFrame kullanılarak yürütülen işlemlerde yaklaşık olarak 2x daha hızlı gerçekleşmiştir.



Şekil 4.6. Sorgu 6 performans karşılaştırma grafiği

Son performans karşılaştırılması yapılacak grafiğimiz Şekil 4.7.'de yer alan koşula bağlı seçim işlemi gerçekleştirdiğimiz sorgumuzdur. Bu karşılaştırmada ilk 6 sorgudaki GPU'nun performansa olan olumlu katkısı yerine GPU DataFrame ile yapılan sorguların sonuçlanması daha uzun sürmüştür. Tartışma bölümünde bu cuDF'in sorgu işlemini neden daha uzun sürede sonuçlandırdığına ilişkin detaylar yer almaktadır.



Şekil 4.7. Sorgu 7 performans karşılaştırma grafiği

4.2. Tartışma

GPU'nun sağladığı paralelleştirme gücünün kullanılarak büyük verilerin analizinde performansa olan etkilerinin incelendiği bu çalışmada genel anlamda performansın artmasına olan olumlu etkileri gözlemlenmiştir.

Özellikle işlemciye ihtiyaç duyulan yoğun hesap gerektiren isteklerin GPU üzerinde yürütülmesi neticesinde GPU'nun hesaplama gücünün performans artışına olumlu yansımaları gözlemlenmiştir.

İlk 10 kayıt seçimin yapıldığı 1. ve 3. Sorgularda tek farklılık birinde tüm alanlar yer alırken, diğerinde belirli alanlar yer almaktadır. Fakat 1. sorguda Spark DataFrame ile gerçekleştirilen sorgu daha kısa sürede gerçekleşirken, 3. sorguda Spark DataFrame yaklaşık olarak 4x daha uzun sürede işlemi tamamlamıştır. Buna karşın Spark SQL ile aynı işlemin gerçekleşme süreleri dikkate alındığında 3. sorgunun daha hızlı bir şekilde gerçekleştiği gözlemlenmiştir.

Büyük farkın gözlemlendiği diğer sorgu ise toplam kayıt sayısının yer aldığı 4. sorgu olmuştur. Bu durumun oluşmasında sütun yönelimli veri formatının kullanılmasının ve seçim yapılırken tüm alanların yer aldığı satırlar yerine GPU DataFrame ile count() fonksiyonun kullanımında sadece ilgili alana ilişkin kayıt sayısının sorgulanmasının önemli etkisi olmuştur.

GPU'nun gücün etkili olarak kullanılabilmesi için uygulamanın GPU'nun yapısına ve işleyişine uygun olarak geliştirilmiş olması gerektiğinden önceki çalışmalar bölümünde de bahsedilmişti. Basit sorgulara ilişkin isteklerin gerçekleşme sürelerinde kayda değer bir farklılık gözlenmezken, hesap gerektiren 7. Sorgu haricindeki tüm sorgularda GPU'nun performans artışına olan etkisi görülmüştür.

GPU SQL, BlazingSQL'in SQL sorgularının cuDF'e uyarlayarak GPU üzerinde yürütülmesini sağladığına giriş bölümünde değinilmişti. Koşula bağlı filtreleme işleminin gerçekleştiği 7. sorgu performans artışının sağlandığı önceki sorgular ile karşılaştırıldığında GPU DataFrame sorgusu çok daha uzun sürede sonuçlanmıştır. Eğer bu durum cuDF'den kaynaklanan bir problem olsaydı aynı farklılığın GPU SQL'de de gözlemlenmesi gerekirdi, fakat aynı sorguda GPU SQL'in performansında

Spark SQL veya Spark DataFrame'e göre ortalama olarak biraz daha hızlı gerçekleştirildiği görülmektedir. Bu durumun oluşmasında "cuDF.query()" fonksiyonunun iyi yapılandırılmamış olması veya cuDF kütüphanesinde yer alan fonksiyonun nasıl kullanılacağına dair verilen cuDF dokümantasyon bilgilerinin yeterli olmamasının etkisi olduğu düşünülmektedir.

5. SONUÇLAR ve ÖNERİLER

5.1. Sonuçlar

Bu çalışmada, büyük verilerin hacim, çeşitlilik ve hız vd. özellikleri, GPU'nun hesaplamadaki gücü ve Apache Spark'ın büyük veri işlemede bellek içi hesaplamadaki başarısını tartışıldı. Ayrıca, iş yükü özellikleri ve donanım yeteneklerindeki mevcut eğilimler ile CPU'nun sistemlerdeki tıkanıklığına bahsedildi. Hem CPU çekirdeğine atanmış gereksiz işlerin azaltılması hem de GPU'nun çalışma şekline uygun olan sütun yönelimli veri formatı kullanıldı.

Büyük verilerin kullanıldığı GPU ile gerçekleştirilen sorguların performans açısından CPU ile yürütülen Spark sorgularına göre işlem gerçekleştirilme sürelerinin gözlemlendiği bu çalışmada, GPU'nun hesaplamalardaki gücünün veri sorgulama sistemlerinde de aynı etkiyi gösterip-göstermeyeceği incelenmiştir.

Kullanıldığı sektöre bağlı olarak farklı amaçlar için tutulan aktivite kayıtlarının analizi, analitiği ve raporlama sürecinin bu verilerin günden güne daha çok ve hızlı artmasına bağlı olarak bu veriler üzerinde yapılacak işlemlerin daha kısa sürede gerçekleştirilmesi için GPU büyük veri sorgulamada kullanılmıştır.

Python dili ile Apache Spark, BlazingSQL ve cuDF kütüphanelerinin kullanıldığı uygulamamızda yaygın olarak kullanılan veri sırlama, filtreleme, gruplama vb. veri tabanı işlemlerinin SQL sorguları ve DataFrame fonksiyonlarının CPU ve GPU kullanılarak ayrı ayrı gerçekleştirilmesi sağlanmıştır.

Uygulamamızda 62.000.000 satırlık 8GB boyutundaki aktivite verisi kullanılarak sorgular hem CPU hem de GPU üzerinde yürütülerek gerçekleştirilmiştir. Aynı sorgular farklı zaman dilimlerinde ve belirli sayılarda gerçekleştirilmiş ve her sorgu ayrı ayrı karşılaştırılmıştır.

Herhangi bir hesaplamanın olmadığı basit sorgularda performans açısından anlamlı bir farklılık gözlemlenmemiştir. Buna karşın hesaplama işlemi gerektiren sorgularda ise GPU üzerinde yürütülen işlemlerde 2x-6x arasında performans artışına bağlı olarak sonuçlar daha kısada sürede elde edilmiştir. GPU'nun ilişkisel

hesaplamalarda var olan paralelleştirme potansiyeli nedeniyle kayda değer bir hızlanma sağlamıştır. Buna karşın işlenecek büyük verinin hacmi gereği bellek tahsisi ve erişimi konusunda GPU'nun yetersiz olduğu görülmüştür.

Sonuç olarak hesaplama gerektiren işlemlerde olduğu gibi veri sorgulamada da GPU'nun kullanılması ile işlemlerin daha kısada gerçekleştirilmesinin mümkün olduğu görülmüştür. Çalışmamızda, sorgu performansını etkileyen GPU bellek ve çekirdek hiyerarşisi ile hesaplama özelliğinin yansımaları özetlemiştir. GPU'lar bulundurdukları çekirdek sayısına göre enerjiyi verimli kullanmaları ve hesaplamalardaki başarısından dolayı iyi bir alternatif olarak değerlendirilebilirler.

5.2. Öneriler

Büyük verilerde hesaplama gücü ve enerjiyi verimli kullanması bakımından GPU'nun kullanılması fikri giderek yaygınlaşmaktadır. Makine öğrenmesi, derin öğrenme, grafik uygulamaları vb. birçok alanda GPU ile başarılı sonuçlar elde edilmiştir. GPU uygulamalarının başarısı uygulanmak istenen algoritmanın GPU'nun işleyişine uygun olarak uyarlanması ile doğru orantılıdır.

Bu çalışmada da görüldüğü gibi yoğun hesaplamaya ihtiyaç duyulan işlemler GPU ile gerçekleştirildiğinde daha iyi sonuçlar elde edilmiştir. Bu durum GPU'nun hesaplamadaki başarısının

GPU belleğinin sabit olması nedeniyle belirli boyuta kadar olan veriler ile çalışılabilmektedir. Daha büyük veriler için aha fazla belleğe ihtiyaç duyulduğundan bellek artırımı yerine çoklu GPU kullanımı ile paralelleştirme gerekmektedir. GPU aygıtı maliyetinin yüksek olması nedeniyle sadece ihtiyaç duyulması durumunda tercih edilebilir çözüm olarak görülebilir.

Hesaplamaların ve bilimsel araştırmaların yoğun olarak kullanıldığı birçok uygulama da GPU kartının kullanımına yönelik destekler bulunmaktadır. Son yıllarda GPU'nun verilerde kullanımına yönelik ciddi bilimsel araştırmalar yapılmaktadır. Yakın gelecekte Hadoop, Spark, Hive vb. büyük ölçekli veri çözümleri bileşenlerinin GPU ile entegre edilmesi sürpriz olmayacaktır.

GPU’lar için hala keşfedilebilecek birçok alan var. Her şeyden önce, bellek erişim modellerini belirleyen ve farklı veri dağılımını idare etmek için yeni yöntemlere ihtiyaç duyulmaktadır. GPU'lar, bellek erişim düzenlerine CPU'lardan daha hassas olduklarından, bu tür yöntemler için algoritma tasarımları ve doğru yöntemlerin seçilmesi çözülmesi gereken önemli problemlerdendir.

Genel olarak, bu çalışmada da görüldüğü gibi GPU’lar veri sorgulamaları için yeni fırsatlar sunmaktadır. Ancak GPU’ların gerçek veri tabanı sistemlerinde kullanılması birçok araştırma ve mühendislik probleminin de çözülmesini gerektirmektedir.

KAYNAKLAR

ABADI, D. J., MADDEN, S., and HACHEM, N., 2008. Column-stores vs. row-stores: How different are they really?. Proceedings of the ACM SIGMOD International Conference on Management of Data, Vancouver, 10-12 June 2008. p967-980.

AHMAD, A., PAUL, A., DIN, S., RATHORE, M. M., CHOI, G. S., and JEON, G., 2018. Multilevel Data Processing Using Parallel Algorithms for Analyzing Big Data in High-Performance Computing. International Journal of Parallel Programming, 46(3): 508–527.

AMD, 2016. AMD A-Series desktop APUs. http://www.amd.com/en-us/products/processors/desktop/a-series-apu, (Erişim Tarihi: 06.03.2016).

ANACONDA, 2016. Visualizing Billions of Data Points: Doing it Right. Anaconda Inc., Austin, Texas, https://know.anaconda.com/On-Demand-Registration-Visualizing-Big-Data-Data-Shader.html, (Erişim Tarihi: 04.08.2019).

APACHE. Apache Spark - Unified Analytics Engine for Big Data. https://spark.apache.org, (Erişim Tarihi: 05.07.2019).

ARMBRUST, M., HUAI, Y., LIANG, C., XIN R., and ZAHARIA, M., 2015. Deep Dive into Spark SQL's Catalyst Optimizer. https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html, (Erişim Tarihi: 21.07.2019)

AWAN, A., IQBAL, M., KHALEEQ,M., and KHALIQ, Y., 2016. Performance analysis of shared-nothing SQL-on-Hadoop frameworks based on columnar database systems. 2016 Sixth International Conference on Innovative Computing Technology (INTECH), Ireland, 24-26 August 2016, p.134-139.

AVCİ SALMA, Ç., TEKİNERDOGAN, B., and ATHANASİADİS, I., 2017. CDomain-Driven Design of Big Data Systems based on a Reference Architecture. In I. Mistrik, R. Bahsoon, N. Ali, M. Heisel, & B. Maxim (Eds.), Software Architecture for Big Data and the Cloud, p49-68.

BAKKUM, P., and SKADRON, K., 2010. Accelerating SQL database operations on a GPU with CUDA. In GPGPU '10: Proceedings of the Third Workshop on General-Purpose Computation on Graphics Processing Units, Newyork p.94-103.

BREß, S., SCHALLEHN, E., and GEIST I., 2013. Towards Optimization of Hybrid CPU/GPU Query Plans in Database Systems. Workshop Proceedings of the 16th East European Conference, Pozna, 17-21 September 2012, p. 27-35.

BOUBELA, R.N., KALCHER, K., HUF, W., NASEL, C., and MOSER, E., 2015. Big Data Approaches for the Analysis of Large-Scale fMRI Data Using Apache Spark and GPU Processing: A Demonstration on Resting-State fMRI Data from the Human Connectome Project. Frontiers in neuroscience, 9: 492p.

CAI, Y., WANG, G., LI G., and WANG, H., 2015, A high performance crashworthiness simulation system based on GPU, Advances in Engineering Software, 86: 29-38.

CANO, J., 2014. The V's of Big Data: Velocity, Volume, Value, Variety, and Veracity. https://www.xsnet.com/blog/bid/205405/the-v-s-of-big-data-velocity-volume-value-variety-and-veracity, (Erişim Tarihi: 25.07.2016).

DATA FLAIR, 2017. Apache Spark Streaming Transformation Operations. https://data-flair.training/blogs/apache-spark-streaming-transformation-operations, (Erişim Tarihi: 20.11.2018).

DAYANANDA, S., 2019a. Spark GraphX Tutorial – Graph Analytics In Apache Spark. https://www.edureka.co/blog/spark-graphx, (Erişim Tarihi: 09.07.2019).

DAYANANDA, S., 2019b. Spark SQL Tutorial – Understanding Spark SQL With Examples. https://www.edureka.co/blog/spark-sql-tutorial, (Erişim Tarihi: 11.07.2019).

DRAKOS, G., 2019. The new era of Huge Data. https://towardsdatascience.com/the-new-era-of-huge-data-16ac4aaaef2b, (Erişim Tarihi: 06.08.2019).

EINAV, L., and LEVIN, J., (2013). The Data Revolution and Economic Analysis. Innovation Policy and the Economy. 14: 1-24.

FAROOQUI, N., ROY, I., CHEN, Y., TALWAR, V., BARIK, R., LEWIS, B.T., SHPEISMAN, T., and SCHWAN, K., 2016. Accelerating Data Analytics on Integrated GPU Platforms via Runtime Specialization. International Journal of Parallel Programming, 46: 336-375.

FAIRCLOTH, J., 2017. Testing enterprise applications. Penetration Tester's Open Source Toolkit (Fourth Edition), Syngress Imprint, p.243-271.

GONZALES, B., 2014. Linking Libraries to the Web: Linked Data and the Future of the Bibliographic Record. Information Technology and Libraries, 33(4): 10-22.

HAYNES, D., RAY, S., MANSON, S.M., and SONI, A. 2015. High Performance Analysis of big Spatial Data. 2015 IEEE International Conference on Big Data, Santa Clara, 29 Oct- 01 Nov 2015, p.1953-1957.

HARRIS, M., 2017. An Easy Introduction to CUDA C and C++. https://devblogs.nvidia.com/easy-introduction-cuda-c-and-c/, (Erişim Tarihi: 27.07.2019).

HE B., Yang, K., Fang, R., Lu, M., Govindaraju, N. K., Luo, Q., and Sander, P., (2008). Relational Joins on Graphics Processors. Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, 10-12 June 2008, p.511-524.

HE, B., LU M., YANG, K., FANG, R., GOVINDARAJU, N.K., LUO, Q., and SANDER, P.V., 2009. Relational query coprocessing on graphics processors, ACM Transactions on Database Systems (TODS), 34(4):1-39.

HEIMEL, M., and MARKL, V., 2012. A First Step Towards GPU-assisted Query Optimization. The Third International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures, Istanbul, 27 August 2012, p.33-44.

HOFMANN, E., 2017. Big data and supply chain decisions: the impact of volume, variety and velocity properties on the bullwhip effect. International Journal of Production Research, 55(17): 5108–5126.

HONGJU, X., FEIW, W., FENMEI, W., and XIUZHEN, W., 2017. Some Key Problems of Data Management in Army Data Egineering Based on Big data. 2017 IEEE 2nd International Conference on Big Data Analysis, Beijing, p.149-152.

HOY, B., 2014. Big data: An introduction for librarians. Medical Reference Ser-vices Quarterly, 33(3): 320-326.

HU, H., WEN Y., CHUA, T., and LI, X., 2014. Toward Scalable Systems for Big Data Analytics: A Technology Tutorial. IEEE, 2: 652-687.

ILIĆ, A., PRATAS, F., TRANCOSO, P., and SOUSA, L., 2011. High-Performance Computing on Heterogeneous Systems: Database Queries on CPU and GPU. High Performance Scientific Computing with Special Emphasis on Current Capabilities and Future Perspectives, IOS Press, p.202–222.

KENDALL, R.A., APRÀ, E., BERNHOLDT, D.E., BYLASKA, E.J., DUPUIS, M., FANN G.I., HARRISON R.J., JU J., NICHOLS J.A., NIEPLOCHA J., STRAATSMA T.P., WINDUS T.L., and WONG A.T., 2000. High Performance Computational Chemistry: An Overview of NWChem a Distributed Parallel Application, Computer Physics Communications. Computer Physics Communications, 128(1): 260-283.

KERR, A., F. DIAMOS, G., and YALAMANCHILI, S., 2009. A characterization and analysis of PTX kernels. P.3-12.

KHAN, N., YAQOOB, I., HASHEM, I. A., INAYAT, Z., ALİ, W. K., ALAM, M., and GANI, A., 2014. Big data: survey, technologies, opportunities, and challenges. TheScientificWorldJournal.

LI, P., LUO, Y., ZHANG, N., and CAO, Y., 2015. HeteroSpark: A Heterogeneous CPU/GPU Spark Platform for Machine Learning Algorithms. Conference: International Conference of Networking Architecture and Storage, Boston, 6-7 August 2015, p.347-348.

LU, M., HE, B., and LUO, Q., (2010). Supporting extended precision on graphics processors. Proceedings of the Sixth International Workshop on Data Management on New Hardware, DaMoN 2010, Indianapolis, 7 June 2010 p.19-26.

MALAKAR, R., and VYDYANATHAN, N., 2013, A CUDA-enabled Hadoop cluster for fast distributed image processing, Parallel Computing Technologies. (PARCOMPTECH) National Conference, Bangalore, 21-23 February 2013, p.1-5.

MANZI, D., and TOMPKINS, D., (2016). Exploring GPU Acceleration of Apache Spark. 2016 IEEE International Conference on Cloud Engineering, Berlin, 4-8 April 2016, p.222-223.

MCDONALD, C., 2015. Using Apache Spark DataFrames for Processing of Tabular Data. https://mapr.com/blog/using-apache-spark-dataframes-processing-tabular-data/, (Erişim Tarihi: 03.08.2019).

MCDONALD, C., 2018. Spark 101: What Is It, What It Does, and Why It Matters. https://mapr.com/blog/spark-101-what-it-what-it-does-and-why-it-matters, (Erişim Tarihi: 04.08.2019).

MENG, X., BRADLEY, J., YAVUZ, B., SPARKS, E., VENKATARAMAN, S., LIU, D., FREEMAN, J., TSAI, DB., AMDE, M., OWEN, S., XIN, D., XIN, R., FRANKLIN, M.J., ZADEH, R., ZAHARIA, M.A., and TALWALKAR, A., 2015. MLlib: Machine Learning in Apache Spark. IEEE International Congress on Big Data (BigData Congress), June 27-July 02 2015, New York, p9-16.

MISHRA, R., and SHARMA, R, 2015. Big Data: Opportunities and Challenges. International Journal of Computer Science and Mobile Computing, 4(6): 27-35.

NEZARAT, A., RAJA, M.M. and DASTGHAYBIFARD G., 2015, A New High Performance GPU-Based Approach to Prime Numbers Generation, World Applied Programming, 5(1): 1-7.

NVIDIA, 2011a. CUDA C Programming Guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, (Erişim Tarihi: 25.08.2019).

NVIDIA, 2011b. Energy Efficiency. http://www.nvidia.com/object/gcr-energy-efficiency.html, (Erişim Tarihi: 15.07.2017).

NVIDIA, 2016a. GPU Programlama. https://www.nvidia.com.tr/object/gpu-programming-tr.html (Erişim Tarihi: 12.04.2018).

NVIDIA, 2016b. CUDA Paralel Hesaplama. http://www.nvidia.com.tr/object/cuda-parallel-computing-tr.html, (Erişim Tarihi: 09.05.2019).

OHNO, Y., MORISHIMA, S., and MATSUTANI, H., 2016. Accelerating Spark RDD Operations with Local and Remote GPU Devices. IEEE 22nd International Conference on Parallel and Distributed Systems, Wuhan, 13-16 December 2017, p.791-799.

OUSSOUS, A., BENJELLOUN, F., LAHCEN, A.A., and BELFKIH, S., 2017. Big Data technologies: A survey. Journal of King Saud University - Computer and Information Sciences, 30(4): 431–448.

OWENS, J., HOUSTON, M., LUEBKE, D., GREEN, S., STONE, J., and PHILLIPS, J., 2008. GPU computing. Proceedings of the IEEE., 96: 879-899.

OPHER , A., CHOU, A., ONDA, A., and SOUNDERRAJANK., 2015. The Rise of the Data Economy: Driving Value through Internet of Things Data Monetization. https://www.ibm.com/downloads/cas/4JROLDQ7, (Erişim Tarihi: 01.08.2019).

PANDEY, S., and TOKEKAR, V., 2014, Prominence of MapReduce in Big Data Processing. 2014 Fourth International Conference on Communication Systems and Network Technologies, 7-9 April 2014, p.555-560.

PAZ, A., and PLAZA, A., 2011. A New Morphological Anomaly Detection Algorithm for Hyperspectral Images and its GPU Implementation. Proceedings of SPIE - The International Society for Optical Engineering, 8157: 1-8.

PLASE, D., NIEDRITE, L., and TARANOVS, R., 2017. A Comparison of HDFS Compact Data Formats: Avro Versus Parquet. Electronics and Electrical Engineering / Elektronika ir Elektrotechnika, 9(3): 267-276.

RAPIDS. https://rapids.ai, (Erişim Tarihi: 03.07.2019).

RATHORE, M.M., SON, H., AHMAD, A., PAUL, A., and JEON, G., 2017. Real-Time Big Data Stream Processing Using GPU with Spark Over Hadoop Ecosystem. International Journal of Parallel Programming, 46:630-646.

RAUHE, H., DEES, J., SATTLER K.U., and FAERBER, F., 2013 “Multi-level parallel query execution framework for cpu and gpu,” in Advances in Databases and Information Systems. Lecture Notes in Computer Science, 8133: 330–343.

RIZATTI, L., 2016. Digital Data Storage is Undergoing Mind-Boggling Growth. https://www.eetimes.com/author.asp?section_id=36&doc_id=1330462, (Erişim Tarihi: 08.07.2019).

SAS, 2016. What is Big Data?. https://www.sas.com/tr_tr/insights/big-data/what-is-big-data.html, (Erişim Tarihi: 09.07.2019).

SALMI, M. F., 2016. Processing Big Data in Main Memory and on GPU. The Ohio State University, Master of Science, 73p.

SHAMSUDDİN, S.M., and HASAN, S. 2015. Data Science vs Big Data @ UTM Big Data Centre. 2015 International Conference on Science in Information Technology (ICSITech), p.1-4.

SHARMA, K., MARJIT, U., and BISWAS, U., 2018. Efficiently Processing and Storing Library Linked Data using Apache Spark and Parquet. Information Technology and Libraries. 37(3): 29-49.

SINGER, G., 2013. The history of the modern graphic processor. http://www.techspot.com/article/650-history-of-the-gpu, (Erişim Tarihi: 09.07.2019).

Spark SQL - Quick Guide, https://www.tutorialspoint.com/spark_sql, (Erişim Tarihi: 11.07.2019).

SINHA, S., 2019. Spark vs Hadoop: Which is the Best Big Data Framework?. https://www.edureka.co/blog/apache-spark-vs-hadoop-mapreduce, (Erişim Tarihi: 09.07.2019).

ŞAHİN, H., KÜLÜR ve M. S., 2016. GPGPU Yöntemi ile Görüntülerin Gerçek Zamanlı Ortorektifikasyonu. Harita Dergisi, 155: 12-22.

TAPDIYA, A., and FABBRI, D. 2017. A Comparative Analysis of state-of-the-art SQL-on-Hadoop Systems for Interactive Analytics. 2017 IEEE International Conference on Big Data, Boston, 11-14 December 2017, p.1349-1356.

TECHVIDVAN TEAM, 2018. Apache Spark Ecosystem Components. https://techvidvan.com/tutorials/spark-ecosystem, (Erişim Tarihi: 12.07.2019)

TUTORIALS POINT. Apache Spark. Tutorials Point Company, INDIA. https://www.tutorialspoint.com/apache_spark/apache_spark_introduction.htm, (Erişim Tarihi: 17.07.2019).

YUAN, Y., LEE, R., and ZHANG, X., 2013. The Yin and Yang of processing data warehousing queries on GPU devices. Proceedings of the VLDB Endowment, 6(10): 817-828.

WANG, D., 2018. Navigating the 3Vs of Big Data: Volume, Velocity, Variety. Laboratory Equipment, 55(4):16–17.

WANG, K., KHAN, and M.M.H., 2015. Performance Prediction for Apache Spark Platform, High Performance Computing and Communications (HPCC), IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), IEEE 12th International Conference on Embedded Software and Systems (ICESS), IEEE 17th International Conference, New York, p.166-173.

WEISS, K., 2019. BlazingSQL in Six Lines of Code. https://blog.blazingdb.com/blazingsql-in-six-lines-of-code-59b88f86580a, (Erişim Tarihi: 01.08.2019).

WEAVER, W., 2014. Need for Speed: Ramping up the Velocity of Big Data. Scientific Computing, p.16–17.

WILLIAMS, J. W., KAREEM, S.A, INTERRANTE, J., MCHUGH, J. and POOL, E., 2015. Bridging High Velocity and High Volume Industrial Big Data Through Distributed In-Memory Storage & Analytics. Proceedings 2014 IEEE International Conference on Big Data, IEEE Big Data 2014, p.932-941.

WINCENT, H., 2017. How many threads can run on a GPU?. https://streamhpc.com/blog/2017-01-24/many-threads-can-run-gpu/, (Erişim Tarihi: 03.08.2019).

WOLF, M., 2014. High-Performance Embedded Computing (Second Edition), Morgan Kaufmann, p.59-138.

WREDE, F., and ERNSTING, S., 2017. Simultaneous CPU–GPU Execution of Data Parallel Algorithmic Skeletons. International Journal of Parallel Programming. 46(1): 42-61

WU, H., 2015. Acceleration and Execution of Relational Queries Using General Purpose Graphics Processing Unit (GPGPU). Georgia Institute of Technology, School of Electrical and Computer Engineering, Doctor of Philosophy, 134p.

WANG, K., ZHANG, K., YUAN Y., MA, S., LEE, R., DİNG, X., and ZHANG, X., 2014. Concurrent analytical query processing with GPUs. Proceedings of the VLDB Endowment, 7(11): 1011-1022.

ZAHARIA, M., CHOWDHURY, M.J., FRANKLIN, M., SHENKER, S., and STOICA, I., 2010. Spark: Cluster Computing with Working Sets. Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. 22-25 June 2010, Berkeley, p.1-10.

ZAHARIA, M., CHOWDHURY, M., DAS, T., DAVE, A., MA, J., MCCAULY, M., FRANKLIN, M.J., and SHENKER, S., STOICA, I., 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. 25-27 April 2012. Sa Jose, p.15-28.

ZAHARIA, M., CHOWDHURY, M., DAS, T., DAVE, A., MA, J., MCCAULEY, M., FRANKLIN, M., SHENKER, S., and STOICA, I., 2012. Fast and interactive analytics over Hadoop data with Spark. Networked Systems, 37(4): 45-51.

ZIKOPOULOS P. C., DEROOS D., PARASURAMAN K., DEUTSCH T., CORRIGAN D. and GILES J., 2012. Harness the Power of Big Data. McGraw Hill Professional, USA, 281p.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Mehmet TURAN
Uyruğu : T.C.
Doğum Yeri ve Tarihi : Adıyaman / 01.03.1984
Telefon : 05464984540
E-Posta : mehmetturan2015@gmail.com

EĞİTİM

Derece	Adı, İl, İlçe	Bitirme Yılı
Lise	: İmam Hatip Lisesi, Adıyaman, Merkez	2001
Üniversite	: Boğaziçi Üniversitesi, İstanbul, Beşiktaş	2012
Yüksek Lisans	: Harran Üniversitesi, Şanlıurfa, Haliliye	2019

İŞ DENEYİMLERİ

Yıl	Kurum	Görevi
2010 - 2014	Mimar Sinan Güzel Sanatlar Üniversitesi	Bilgisayar İşletmeni
2014 - devam	Adıyaman Üniversitesi	Öğretim Görevlisi

YABANCI DİLLER

Sınav	Tarih	Sonuç
YökDil (İngilizce)	10.03.2019	86,25
YDS (İngilizce)	24.03.2019	57,25

YAYINLAR

TURAN, M., NACAR, M. A., 2016. Asal Sayıların Eratosten Kalburu Algoritması Kullanılarak Bulunmasında GPU ve CPU Başarımlarının Analizi. Adıyaman Üniversitesi Mühendislik Bilimleri Dergisi, 3(5):71-76