MEKANSAL BÜYÜK VERİ KÜMELEME

Yağmur KILIÇ

Yüksek Lisans Tezi
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Yrd. Doç. Dr. Galip AYDIN

EYLÜL-2015

T.C
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MEKANSAL BÜYÜK VERİ KÜMELEME

YÜKSEK LİSANS TEZİ

Yağmur KILIÇ

121129110

Tezin Enstitüye Verildiği Tarih : 16 Eylül 2015
Tezin Savunulduğu Tarih : 01 Ekim 2015

Tez Danışmanı : Yrd. Doç. Dr. Galip AYDIN

Diğer Jüri Üyeleri : Prof.Dr. Mehmet KAYA

Prof.Dr. Ali KARCI

EYLÜL-2015

ÖNSÖZ

Bu tez çalışmasında büyük miktardaki mekânsal verinin işlenmesini sağlayacak dağıtık bir mimari kurulmuştur. Açık kaynak Bulut Bilişim yazılımları ile kurulan bu mimari üzerinde yine açık kaynaklı Büyük Veri işleme ve analizi yazılımları kurulmuştur. Bu sistem kullanılarak Veri Madenciliğinin önemli tekniklerinden birisi olan K-Means kümeleme algoritması ile milyonlarca noktasal verinin kümelenmesi sağlanmıştır. Tezde kullanılan teknolojilerin ve yazılımların kurulum adımları ve testlerin gerçekleştirilme adımları benzer çalışmalara referans olabilmesi için detaylı bir şekilde yazılmıştır.

Bu çalışma boyunca yardımlarını esirgemeyen danışman hocam Yrd. Doç. Dr. Galip AYDIN'a teşekkürlerimi sunarım. Ayrıca hayatımın her anında ilgi, anlayış ve desteklerini esirgemeyen aileme çok teşekkür ederim.

TEŞEKKÜR

Bu tez çalışması TÜBİTAK 2210-C Öncelikli Alanlara Yönelik Yurt İçi Yüksek Lisans Bursu ile desteklenmiştir. Tezdeki yazılımsal ve donanımsal uygulamaların temin edilmesinde maddi desteklerinden dolayı Türkiye Bilimsel ve Teknolojik Araştırma Kurumuna teşekkür ederim.

İÇİNDEKİLER

ÖZET

Konum verisi üreten GPS cihazlarının ve akıllı telefonların yaygınlaşması ile Dünya çok büyük miktarda mekânsal veri üretilmektedir. Bu verilerin işlenebilmesi ve analiz edilebilmesi için geleneksel veri analiz çözümleri yetersiz kaldığından son yıllarda alternatif çözümler aranmaya başlanmıştır. Büyük veri teknolojileri olarak adlandırılan yeni yaklaşımlar çok büyük miktardaki veriyi dağıtık olarak ve yüksek performansla analiz edebilmemize yardımcı olmaktadır.

Bu tez çalışmasında mekânsal verilerin analizi ve kümelenmesi için Büyük Veri teknolojilerinin kullanımı incelenemiş, milyonlarca noktanın kısa zamanda kümelenebilmesi için açık kaynak Büyük Veri teknolojileri olan Apache Hadoop ve Apache Mahout kullanımı gerçekleştirilmiştir.

Dağıtık Büyük Veri analizi sistemlerinin paralel olarak analiz yapabilmesi için gerekli olan sunucular açık kaynak OpenStack Bulut Bilişim altyapı yazılımı ve Google Cloud kullanılarak sağlanmıştır.

SUMMARY

SPATIAL BIG DATA CLUSTERING

In recent years widespread use of GPS devices and the smart phones resulted in the generation of big amounts of spatial data.As the traditional data analysis methods can not cope with such big data problems, alternative solutions are being researched. So called Big Data Technologies help us analyze the big data in a distributed and high performance fashion.

In this thesis use of Big Data Technologies for analyzing spatial data is investigated and for clustering millions of points open soırce Big Data Technologies Apache Hadoop and Apache Mahout are employed.

The servers required for the Distributed Big Data analysis system to realize parallel analysis are created using open source OpenStack Cloud Computing software and Google Cloud.

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

SEMBOLLER LİSTESİ

CBS / GIS **:** Coğrafi Bilgi Sistemi

GFS **:** Google File System

GPS **:** Küresel Yer Belirleme Sistemi ya da Küresel Konumlandırma Sistemi

HDFS **:** Hadoop Distibuted File System

LHC **:** Büyük hadron çarpıştırıcısı

SKA **:** Kilometre Kare Dizgesi- Square Kilometer Array

VTBK **:** Veri madenciliği veri tabanlarından bilgi keşfi

VTYS **:** Veri tabanı yönetim sistemleri

1. GİRİŞ

Mobil cihazlar, akıllı telefonlar, GPS cihazları vb konum tespit eden ve bu konumları paylaşabilen teknolojilerin gelişmesi ve bunların daha önce görülmemiş bir hızla yaygınlaşmaları, çok büyük miktarda konum verisinin üretilmesi sonucunu doğurmuştur. Milyarlarca cep telefonunun ve milyonlarca GPS alıcısı cihazın ürettiği bu devasa miktardaki veriler farklı amaçlarla kullanılmakta olsa da genelde sadece anlık olarak kullanılır ve bir süreliğine depolanırlar.

Son yıllarda mekânsal verinin içerisinde aslında çok kıymetli, gizli bilgiler ve örüntüler de depoladığı düşünülerek bu veriler üzerinde farklı analizler gerçekleştirilen çeşitli çalışmalar yürütülmüştür. Örneğin bir cep telefonundan elde edilen konum verilerinin düzenli olarak toplanması ve bunların doğru yöntemlerle analizi, telefonun sahibinin seyahatleri, düzenli ziyaret ettiği mekanlar, rutin alışkanlıkları vb bilgilere ulaşmamızı sağlayabilir. Dahası, çok sayıda insanın konum verilerinin beraber incelenmesi, bu insanların ortak hareket alışkanlıkları, aynı mekanlarda bulunma sıklıkları, arkadaşlıklar, sosyal ilişkiler gibi gizli bazı bilgilerin çıkarılmasına da yardımcı olabilir.

Ancak bu tür analizler yapabilmek için iki temel problemin çözülmesi gerekmektedir:

1- Mekânsal verilerden gizli örüntülerin çıkarılması için kullanılacak yaklaşımların belirlenmesi,

2- Çok büyük verilerin analizi için çözüm bulunması.

Bilim insanları ve şirketler mekânsal verinin analizini gerçekleştirmek için istatistik, Veri Madenciliği ve Makine Öğrenmesi algoritmalarını uzun yıllardır kullanmaktadırlar. Ancak sosyal medyanın yaygınlaşması ile konum paylaşımının da yaygınlaşması, akıllı telefonlar ve mobil cihazların konum bilgisini depolayabilmeleri ve çeşitli uygulamalarla paylaşabilmeleri ile daha önce görülmemiş sayılarda veri üreticisinin ortaya çıkmasını ve sadece 10 yıl öncesi için hayal edilemeyecek boyutlarda konum verisinin anlık olarak üretilebilmesini sağlamıştır. Dolayısıyla geleneksel istatistiki modellerin veya Veri Madenciliği algoritmalarının bu kadar yüksek boyutlarda veriden sonuç elde etmeleri imkansız hale gelmektedir.

Veri Madenciliği yöntemleri ile mevcut veri yığınları içerisinden anlam ifade eden örüntülerin çıkarılması mümkün olmaktadır. Ancak geleneksel algoritmalar göreceli küçük

veri setleri üzerinde çalışmak için geliştirilmiştir. Çok büyük veri setlerinin işlenebilmesi için Paralel Hesaplama gibi bilimsel yaklaşımlar da ortaya konmuş ancak genellikle yüksek oranda uzmanlık ve özel donanımlar gerektiren bu yaklaşımlar son kullanıcıya veya analizciye hitap etmemiştir.

Son yıllarda hızlı bir yayılma ve popülerleşme eğilimi gösteren Büyük Veri yaklaşımları bu tür problemlerin çözümünde bize yardımcı olabilmektedir. Ortalama özellikli donanımların özel orta katman yazılımları ile bir araya getirilmesini sağlayan ve çok basit programlama arayüzleri ile Dağıtık-Paralel hesaplama imkanı sunan Büyük Veri teknolojileri, çok büyük miktardaki mekânsal verilerin analizinde de kullanılabilir. Böylece örneğin geleneksel yöntemlerle en fazla birkaç milyon nokta içerisindeki kümeleri tespit etmek mümkün iken Dağıtık-Paralel Büyük Veri yaklaşımları ile milyarlarca noktayı analiz etmek ve kümeleri bulmak mümkün olabilir.

Bu tezde, çok büyük miktardaki konum verisi içerisinden veri madenciliği yöntemleri ile örüntülerin çıkarılması için geliştirilen, Bulut altyapısı üzerinde çalışan ve Büyük Veri teknolojileri kullanan Dağıtık Mekansal Veri Madenciliği Sistemi sunulmuştur.

Tezin 2. Bölümünde sistemde kullanılan Büyük Veri, Bulut Bilişim, MapReduce, Hadoop gibi teknolojilerden ve ilgili konulardan kısaca bahsedilmiş ve bunların sistemde nasıl kullanıldığı açıklanmıştır. Tezin konusu olan mekânsal verilerle ilgili kısa bir açıklama verilmiş, GPS gibi teknolojilerle toplanan mekânsal verilerin yaygın formatları ve kullanılan koordinat sistemleri açıklanmıştır.

3. Bölümde Veri Madenciliği tartışılmış ve Dağıtık Veri Madenciliği kısaca sunulmuştur. Aynı zamanda Mekansal Veri Madenciliğinin tanımı, kullanılma sebepleri ve ilgili çalışmalar verilmiştir. Mekansal veri madenciliğinin çok büyük verilerle kullanılabilmesi için bu algoritmaların nasıl dağıtık çalıştırılacağı da tartışılmıştır.

Tezin 4. Bölümünde ise geliştirilen sistem üzerinde çalıştırılan Dağıtık Mekânsal Veri Madenciliği uygulamaları sunulmuştur. Uygulama sonuçları verilerek tezin elde ettiği çıktılar ayrıntılı bir şekilde tartışılmıştır.

1.1. Literatür Özeti

Literatürde Mekânsal veri madenciliği ve Makine Öğrenmesi ile ilgili çok sayıda çalışma bulunmaktadır. Bu çalışmalar genel olarak mekânsal referans taşıyan verilerin sınıflandırılması, kümelenmesi veya veri içindeki örüntülerin bulunmasına odaklanmıştır. Ancak literatür taramasında dağıtık mekânsal veri madenciliği konusunun oldukça yeni olduğu görülmektedir. Büyük miktardaki mekânsal verinin paralel olarak işlenmesi ile ilgili yapılmış çalışmalar olsa da bunlar genelde büyük projeler kapsamında gerçekleştirilmiş yüksek seviyeli bilimsel çalışmalarla ilişkilidir. Büyük Mekansal Veri ile ilgili çok az sayıda çalışma olduğu, Büyük Veri teknolojileriyle Büyük Mekansal Veri analizi konusunun henüz yeni olduğu görülmüştür. Aşağıda mekânsal veri analizi ile ilgili yapılan bazı yayınlar özetlenmiştir.

İnsanlar arasındaki ilişkilerin keşfi için GPS verileri kullanılmıştır. İnsanlar arasındaki ilişkinin seviyesini anlamak için GPS verilerinden alınan zaman ve mekan bilgilerinin kullanılabileceği [1] nolu makalede öne sürülmektedir. İnsanların aynı yerlerde, aynı zamanlarda bulunması, benzer yerleri ziyaretleri, aynı insan gruplarıyla iletişim kurmaları aynı sosyal ağ içerisinde olduklarını gösterebilir. Bu makale GPS verilerinin RDF ve Veri Madenciliği ile beraber kullanılarak ilişkilerin tespit edilebileceğini tartışmaktadır. İlişkileri keşfetmek için Binning ve Birliktelik Kuralları kullanılmıştır. Sistemde mekanlar ve kişiler takip edilmektedir. Her bir mekan için bir benzersiz no kullanılmaktadır. Bir mekanı ziyaret eden kişi ve ziyaret zamanı o mekanın listesine eklenmektedir. Böylece verilen bir zaman dilimi içerisinde belirlenen bir mekanı ziyaret eden kişilerin listesine ulaşılabilmektedir. Bu veriler elde edildikten sonra veri madenciliği kullanılarak kişiler arasındaki ilişkiler belirlenmeye çalışılmıştır.

GPS verileri kullanılarak insanlar arasındaki ilişkiler iki boyutta belirlenebilir: Spatial Locality (mekansal yakınlık) ve Temporal Locality (zamansal yakınlık). Mekansal yakınlık prensibi aynı mekanlarda bulunmuş kişilerin, zamansal yakınlık ise aynı zamanda aynı yerde bulunmuş kişilerin ilişkili olduklarını öngörür.

Belirli bir zaman ve mekanda verilen bir noktada, varsa, ajanların art arda ziyaret ettiği dairesel, eliptik veya dikdörtgensel bölge neresidir sorusuna cevap verebilmek için [2] nolu

makalede bir metot sunulmuştur. Bu makalede, Kuldorff tarafından önerilen uzay-zaman istatistik yaklaşımını geliştirmiş ve metropolitan alanlarda toplanan GPS noktalarına uygulamışlardır. CitySense amaç olarak uzaysal ve uzay-zaman kümelerini bularak bunların tesadüfi mi yoksa istatistiki olarak önemli olup olmadığını anlamaya ve eğer gerçek kümeler ise ilgili kişilerin daha sonra ziyaret edebilecekleri yerleri tahmin etmeye çalışmaktadır. CitySense bir şehir için hot-spot tespiti yapmaktadır. Bunun için tamamen zamansal (purely temporal) tek boyutlu, tamamen uzaysal (purely spatial) tek, iki veya üç boyutlu ve uzay-zaman (iki-dört boyutlu) kümelerin tespiti için kullanılır.

Kişinin günlük hayatında anlam ifade eden yerlerin konumlarını ve etiketlerini elde etmeye [3] nolu makalede çalışılmıştır. Önemli yerler olarak adlandırılan bu konumlar insanların belirli bir rutin içerisinde ziyaret ettiği yerlerin birçok farklı nokta arasından çıkarılması ile bulunmaktadır.

Foursquare check-in verileri kullanılarak, bir şehirde insan gruplarının gittikleri yere göre tanımlanması [4] nolu makalede yapılmıştır. Bu şekilde insanlar ve mekanlar arasındaki ilişkilerden hareketle sosyal bağlantıların tespiti yapılmaktadır. Bu tür bir veri arkadaş tavsiyeleri, benzer eğilimlere sahip insanlara uygun tavsiyelerde bulunulması amacıyla kullanılabilir.

Yukarıda özetlenen bu çalışmaların dışında birçok çalışmada da [5-14] benzer veya farklı konum tabanlı analiz çalışmaları yapılmıştır.

2. ARKA PLAN VE İLGİLİ TEKNOLOJİLER

Bu çalışma Büyük Mekânsal Veri (Big Spatial Data) olarak isimlendirilen ve geleneksel yöntemlerle analiz edilemeyecek kadar büyük miktardaki mekânsal verinin, Veri Madenciliği yöntemleri ile analiz edilebilmesi için, Büyük Veri teknolojilerini kullanan bir sistem kurmayı ve örnek bazı analizler yapmayı hedeflemiştir.

Dolayısıyla tezin bu bölümünde, kurulması hedeflenen Büyük Mekânsal Veri Analiz Sistemini oluşturan teknolojiler ve bu teknolojilerin arka planındaki önemli bazı bilimsel konular kısaca açıklanmıştır.

Büyük Veri teknolojileri kısaca, akademik dünyada Dağıtık Sistemler ve Paralel Programlama olarak bilinen bilimsel alanların beraber kullanılması ve bunların modern bilişim teknolojileriyle birleşimi sonucunda ortaya çıkmıştır. Dağıtık Sistemler kısaca fiziki olarak bağımsız ve birbirinden farklı lokasyonlardaki hesaplama kaynaklarının, büyük problemlerin çözülmesi için ortak olarak kullanılmasını sağlayan sistemlere verilen ortak isimdir.

Paralel hesaplama pahalı hesaplama kaynaklarının eş zamanlı olarak (koşut) ço sayıda iş yapabilmesi için geliştirilmiş çok sayıda yaklaşım, yazılım ve algoritmalara verilen genel bir isimdir.

Büyük Veri, işte bu iki önemli bilimsel alanın birleştirilerek, çok sayıda dağıtık makine üzerinde, paralel olarak hesaplama ve işlem yapılmasını ve böylece zaman ve maliyetten tasarruf etmeyi amaçlamaktadır. Büyük Veri teknolojileri, geleneksel yollarla işlenemeyecek kadar büyük miktardaki verilerin işlenmesi için pratik ve düşük maliyetli çözümler sunduğu için de çok hızlı bir şekilde kabul görmüş ve popülerleşmiştir.

Bu bölümde öncelikle Dağıtık Sistemler daha sonra da Paralel Hesaplama kavramları kısaca açıklanmıştır.

Bilgisayarlar, sunucular, veri depolama sistemleri genel bir isimlendirme ile hesaplama kaynağı olarak adlandırılmaktadırlar. Bu kaynaklar yüksek maliyetli olduğundan, bilgisayar bilimcileri, mümkün olduğunca çok sayıda problemi, mümkün olduğunca az sayıda hesaplama kaynağı kullanarak çözmeye çalışmışlardır. Bu amaçla geliştirilen bir teknoloji olan sanallaştırma, bir sunucu bilgisayar üzerinde birden fazla işletim sistemini eş zamanlı olarak ve birbirinden bağımsız bir şekilde çalıştırmaya yarayan bir teknolojidir. Sanallaştırma teknolojisinin olgunlaşması, eş zamanlı olarak web servislerinin

yaygınlaşması ve ağ teknolojilerindeki gelişmelerle beraber, donanımsal kaynakların da servis olarak kullandırılabileceği fikri oluşmuştur. Bulut Bilişim işte bu amaca yönelik olarak geliştirilmiş popüler bir teknolojiler bütünüdür. Bulut Bilişimin türlerinden birisi olan IaaS (Infrastructure as a Service), altyapının hizmet olarak kullandırılmasını, diğer bir ifadeyle sunucuların sanallaştırılmasının web servisleri gibi API'ler kullanılarak yapılmasını sağlamaktadır. Bu tez çalışmasında açık kaynaklı bir Bulut Bilişim yazılımı kullanılarak sanal makineler oluşturulmuş ve bu sanal makineler üzerinde oluşturulan bilgisayar kümesi (cluster) üzerinde Büyük Veri analiz yazılımları çalıştırılmıştır. Bu sebeple de tezin bu bölümünde Bulut Bilişim kavramı açıklanmıştır.

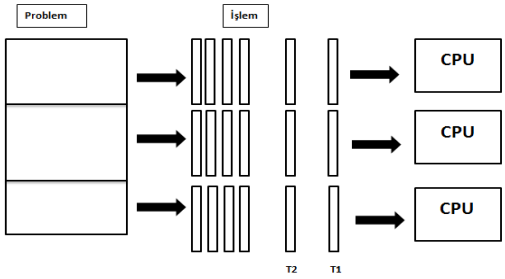
Bölümün son kısmında ise Büyük Veri kavramı açıklanmış, Büyük Veri teknolojilerinin çok kullanılanları olan Hadoop, HDFS, Google File System, Apache Spark ve Apache Mahout anlatılmıştır.

2.1. Paralel Hesaplama

Hızla gelişmekte olan bilgisayar teknolojisi yazılımların ihtiyaçlarını karşılamakta zorlanmaktadır. Daha büyük hafızalı, daha hızlı bilgisayarlara olan ihtiyaç her geçen gün artmıştır. Bilim insanları karmaşık problemlerin çözümünde özel bazı bilgisayar mimarileri kullanmışlardır. Bu tür problemler için bulunan çözümlerden birisi, birden fazla CPU barındıran özel bilgisayar mimarileri geliştirmek olmuştur. Böylece aynı anda birden fazla iş eş zamanlı olarak farklı CPU'lar üzerinde çalıştırılabilmektedir. Ancak bu özel mimariler veya daha fazla donanımsal kaynak kullanımı maliyetleri çok büyük oranda arttırdığından bilim insanları mevcut kaynakların daha verimli bir şekilde kullanılabilmesinin yollarını aramışlardır. Bu amaçla da Paralel Programlama geliştirilmiştir. Çünkü paralel işlemci mimarisini daha etkili bir şekilde kullanmak için aynı zamanda yazılımın da paralel olarak programlanması gerekmektedir. Bu amaçtan dolayı yoğun olarak kullanılan PVM (Paralel Virtual Machine) [15], endüstri standardı olan MPI (Message Passing Interface) [16,17,18] gibi paralel programlama kütüphaneleri geliştirilmiştir.

Bilgisayar programları, seri veya paralel olarak yürütülebilirler. Seri programlamada bir işlem tek bir işlemci üzerinde yaptırılırken paralel programlamada aynı işlem parçalara ayrılarak, farklı işlemciler veya farklı çekirdekler üzerinde yürütülür. Bu işlemciler aynı

makine üzerinde olabileceği gibi bir ağa bağlı bilgisayarlarda da olabilmektedir. Böylece işlem süreçleri işlemciler arasında paylaşıldığından işlem süresi de kısalır [5-6].



Şekil 2.1. Paralel programların çalışması

Paralel hesaplama, genel olarak Dağıtık hesaplamadan farklı olarak, çok işlemcili özel tasarım gerektiren donanımlar üzerinde yapıldığından maliyeti daha yüksektir.

2.2. Dağıtık Sistemler

Dağıtık sistemler birbirinden bağımsız bilgisayarların oluşturduğu, kullanıcıya tek bir bilgisayarmış gibi görünen, haberleşmenin mesajlaşma sayesinde gerçekleştirdiği bir ağ olarak tanımlanır [19]. Kullanıcı sisteme girdiğinde arka planda tek bir işlemci ve ara yüz görse de dağıtık sistemler, farklı bilgisayar sistemleri üzerindeki işlemleri bir bütün olarak işleyebilir ve çalıştırabilirler. Dağıtık sistemlerde bir hata oluştuktan sonra işleme devam edilebilir, kaynakların paylaşımı yapılabilir, farklı sağlayıcılardan gelen yazılım ve donanımlar kullanılabilir, eş zamanlı işleme sayesinde performans arttırılabilir. Dağıtık sistemler merkezi sistemlere göre karmaşık bir yapıya sahiptir ve bu yüzden sistem yönetimi kısmında daha çok uğraş gerekir. Çoklu işlemci mimarileri arasında en basit dağıtık sistem modelidir [20].

Dağıtık sistem, Şekil 2.1'de görüldüğü gibi bir ağ ile birbirine bağlanmış bağımsız bilgisayarlardan oluşur. Bu bilgisayarlar verilen bir problemin veya verinin bir kısmını kendi üzerlerinde çözecek algoritmalar çalıştırırlar. Sistemin yönetiminden sorumlu yazılım problemlerin dağılımı ve çözüm parçalarının bir araya getirilmesinden ve kullanıcıdan bu detayların gizlenmesinden sorumlu olurlar.. Dağıtık sistemlere, bilgisayar ağları, noktadan

noktaya (peer-to-peer) modelleri, sunucu-istemci sistemleri ve internet örnek olarak gösterilebilir [19].

Tablo 2.1. Dağıtık Sistemlerin Avantajları – Dezavantajları Tablosu

Avantajları	**Dezavantajları**
• Maliyet	• Merkezi yapı zorunluluğu
• Erişim kolaylığı	• Her dilde yazım desteği yok
• Hesaplama ve depo alanı	• Güvenlik Problemleri
• Ölçeklenebilir	• İletişim
• Güvenilir	
• Dağıtım kolaylığı (Web)	
• Güvenlik	

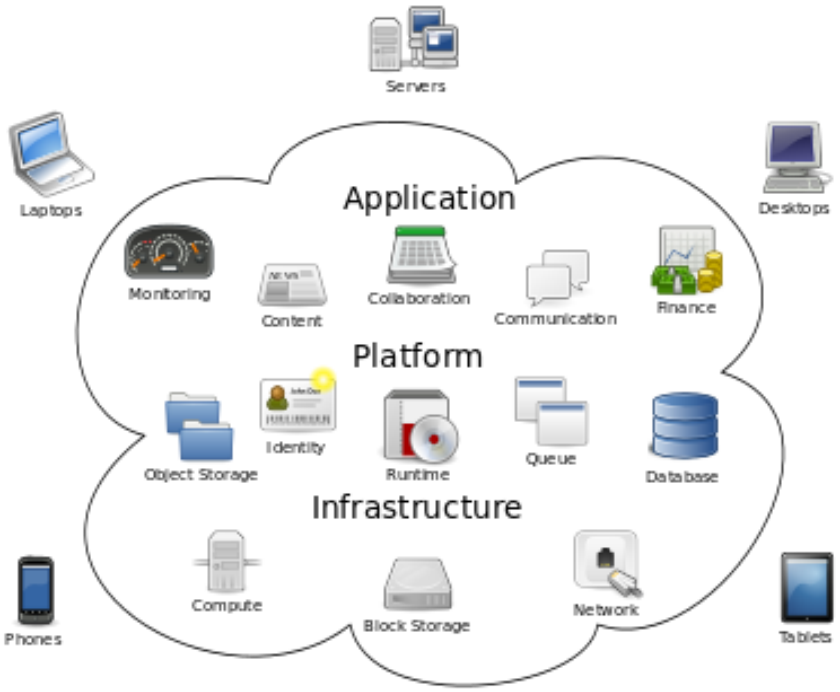
2.3. Bulut Bilişim

Oldukça farklı şekillerde tanımlanan [21-24] Bulut Bilişim kısaca paylaşılan hesaplama kaynaklarına her yerden, kolay ve anında erişim sağlayan bir dağıtık sistem modeli olarak tanımlanabilir. Bulut Bilişim altyapıları, hesaplama kaynağına ihtiyaç duyan işletmelere web üzerinden erişim sağlarlar ve böylece geleneksel satın alma ve işletme modelinden uzaklaşmalarına olanak sağlarlar [25-29]..

NIST tanımına göre Bulut Bilişim, konfigüre edilebilir hesaplama kaynaklarının oluşturduğu paylaşımlı bir kaynak havuzuna ağ üzerinden erişim sağlayan bir bilişim mimarisidir [22].

Bulut Bilişim yaklaşımında, farklı lokasyonlardaki fiziksel sunucular bir yazılım ile birbirine bağlanır ve oluşturulan bu dağıtık altyapı üzerinde ihtiyaç duyulduğu kadar sanal makineler başlatılır. Benzer şekilde depolama sistemleri de müşteriler arasında dağıtılır. Müşterilerin birbirinin kullandığı sanal makineler ve depolama kaynaklarından haberdar olmaması sağlanarak çok kiracılı (multi-tenant) yapı elde edilir. Sanal makineler eldeki kaynaklar sınırında olmak üzere ihtiyaç duyulduğu kadar arttırılabilir, azaltılabilir [29].

Bulut Bilişim hizmet sağlayıcıları tarafından sunulan ve genelde ticari amaçlar için kullanılan bu hizmetler aynı zamanda bilim dünyasının da ilgisini çekmiş ve birçok farklı araştırma projesinde kullanım alanı bulmuştur [30]. OpenStack [32], OpenNebula [33], Nimbus [34] gibi açık kaynak Bulut bilişim yazılımları birçok bilimsel çalışmada sanal makineler ve depolama alanları gibi hesaplama kaynaklarını sağlamak amacıyla kullanılmıştır.



Şekil 2.2. Bulut Bilişim metaforu (https://en.wikipedia.org/wiki/Cloud_computing)

Bulut sistemleri müşterilerin hizmetleri kullanım miktarlarını ve sürelerini tutmakta ve buna göre ücretlendirebilmektedirler.

Bulut Bilişim hizmetleri Altyapı Hizmeti (Infrastructure as a Service - IaaS), Software as a Service - SaaS) ve Platform Hizmeti (Platform as a Service - PaaS) olarak 3 şekilde sunulmaktadır:

2.3.1. Altyapı Hizmet Modeli (IaaS - Infrastructure as a Service)

Bulut Bilişimin en temel hizmetlerinden biri olan Altyapı Hizmet Modeli, kaynakların sanallaştırılmasını, web arayüzleri ve yazılım API'leri aracılığıyla yaptırarak altyapıyı hizmet olarak sunan bir hizmet türüdür. Kullanıcı bu arayüzler ile sanal makine oluşturmak, veri depolamak veya istediği işletim sistemini kullanmak gibi işlemleri uzaktan gerçekleştirebilir. IaaS ile kullanıcılara özel sanal makineler tahsis etmek ve kullanıcıların ihtiyacına göre otomatik kaynak artırımı yapmak mümkün olabilmektedir.

Bulut bilişimde tüm altyapı hazır halde olduğu için zamandan tasarruf sağlanır ve "kullandığın kadar öde" esaslı bir çalışma yapısı sunulur.

Bulut Bilişim sunucu kiralanması, sanal sunucu kiralanması veya hosting hizmeti sunulması gibi geleneksel bilişim hizmetlerinden farklı olarak kullanıcılarına istedikleri zaman, istedikleri özelliklere sahip ve istedikleri sayıda sunucu oluşturabilme ve sadece birkaç saniye içerisinde seçilen işletim sistemi kullanılarak başlatabilme yeteneği sağlar. Sunucu veya depolama sistemi gibi hizmetler aktif oldukları sürece ücretlendirilirler. Bu durum geleneksel sunucu barındırma veya kiralama yaklaşımlarına göre önemli maliyet avantajları sunar.

2.3.2. Yazılım Hizmet Modeli (SaaS - Software as a Service)

Web Servisleri ile yaygınlaşan, programların ve uygulamaların, uzak kullanıcılara standart arayüzler vasıtası ile kullandırılması yaklaşımı Bulut Bilişimin SaaS modelinin temelini oluşturmuştur. Web Servislerinde, programlar SOAP/REST gibi standart yöntemlerle kullandırılırken, SaaS modelinde komple yazılımlar ve paket programlar Bulut üzerinden kullandırılabilmektedir. Müşteriler kullandıkları yazılımın kendileri ile ilgili konfigürasyonlarını değiştirebilir ve özelleştirebilirler.

2.3.3. Platform Hizmet Modeli (PaaS - Platform as a Service)

Platform hizmeti modeli kullanıcılara projelerini çalıştırabilecekleri bir platform ve uygulama geliştiricilere donanım ve yazılım katmanları sunarak geliştirme imkanı sağlanır. Bu hizmet ile, sistem yönetimi, işletim sistemi, programlama dili ortamı, veri tabanı vs. gibi platformlar sunulabilir.

Bu modele Microsoft Azure , IBM Bluemix, Amazon ,Google App Engine örnek olarak verilebilir. Ayrıca Google Cloud tarafından sunulan Büyük veri analiz servisler de platform örneği oalrak düşünülebilir.

Bulut Platform Hizmetleri ile kullanıcılar uygulama geliştirme, test etme ve kullanıma alma süreçlerini hızlandırabilir, herhangi bir altyapı yatırımı yapmadan ve çok sayıda yazılım kurulumuna gerek kalmadan uçtan uca geliştirme süreçlerini yönetebilirler.

2.4. Büyük Veri

İnternetin gelişimi ve yaygınlaşması insanlık tarihinde görülmemiş ölçekte veri üretilmesi ve paylaşılması sonucunu doğurmuştur. Dünya çapında veri depolama

sistemlerinin kapasitesinin her 40 ayda bir ikiye katlandığı belirtilmektedir. [35] 2012 itibariyle her gün 2.5 exabyte (2.5×1018) veri üretildiği belirtilmiştir [36].

Bu ölçekteki verilerin işlenmesi için gerekli kaynakların miktarını arttırmak tek başına bir çözüm değildir. Çünkü hesaplama kaynakları hiçbir zaman veri miktarı kadar hızlı bir gelişme gösteremez. Kaynakların verimli kullanılması ve bu kaynakların kolayca kullanabilmesi ve erişilebilmesi de önemlidir.

Son yıllarda veri üretim hızının tahminlerin ötesinde büyüme ile gerçekleştiği görülmektedir. Bu durum, belirli süreler depolanması gereken ancak gerçekte çok az bir kısmı kullanılan veya işe yarayan, büyük veri depoları ortaya çıkmasına sebep olmuştur. Ancak depolanan ama kullanılmayan veri sadece maliyet oluşturur ve herhangi bir fayda sağlamaz. Zamanla, araştırmacılar arasında aslında bu verilerin içerisinde çok önemli bilgilerin saklı olduğu ve bu bilgilerin, veri işleme ve analiz yöntemleri ile çıkarılabileceği düşüncesi yaygınlaşmış ve bu yönde çalışmalar hız kazanmıştır.

Büyük miktardaki verinin geleneksel yöntemlerle hızlı ve efektif bir şekilde işlenemeyeceği açıktır. Büyük veri içerisindeki örüntülerin çıkarılması, verinin anlamlandırılabilmesi için özel çözümler gerekir.

Geleneksel yöntemler, yaklaşımlar, hesaplama yöntemleri ve algoritmalarla işlenemeyecek kadar çok miktardaki veriye Büyük Veri (Big Data) denir. Büyük Veri kavramı aynı zamanda çok miktardaki verinin işlenmesinde kullanılan teknolojileri de ifade etmektedir.

Bugün özellikle dağıtık ve paralel hesaplama teknikleri ile geliştirilmiş çeşitli veri analiz yöntemleri Büyük Veri analizinde kullanılmaktadır. Bu tez çalışmasında Mekansal Büyük Veri analizi için Büyük Veri yöntemleri incelenmiş ve kullanılmıştır.

Genel anlamda veriler, yapılarına göre iki kısımda incelenebilir:

1. Yapılandırılmış veriler (Structured Data): Belirli bir kurala göre oluşturulmuş ve biçimlendirilmiş verilere denir. Genelde bir şemaya uygun olarak veritabanlarında saklanırlar. Örneğin e-devlet üzerindeki veriler, faturalar, sınav sonuçları vb.
2. Yapılandırılmamış veriler (Unstructured Data): Herhangi bir biçime bağlı olmadan, rasgele üretilen ve farklı formatlarda saklanan verilerdir. Veritabanları dışında, dosyalarda

saklanan veriler olarak düşünülebilir Örneğin Facebook, Twitter, Foursquare vb sosyal medya ortamlarınca üretilen veriler vb.

Dünya çapında mevcut verinin çok önemli bir kısmının yapılandırılmamış olduğu tahmin edilmekte ve farklı çalışmalar bu tür veri miktarını %70 ile %90 arasında olduğunu belirtmektedir. Dolayısıyla yapılandırılmamış verilerle ilgili gerçekleştirilecek çalışmalar büyük bir önem arz etmektedir.

İlişkisel veri tabanlarında tutulan yapısal veriler sorgulanabilir, analiz edilebilir ve sonuç çıkarmak için kullanılabilir. Ancak yapısal olmayan verilerin sorgulanması ve karar destek için kullanılması çok zordur. Yakın zamana kadar bu tür verilerin önemli bir kısmı kullanılamamaktaydı. Büyük Veri teknolojileri ile yapısal olmayan verilerden de kullanılabilir, önemli ve yararlı bilgilerin elde edilebileceği görülmüştür.

Elde edilen bilgiler ilişkisel veri olmadığı için ilişkisel veri tabanları büyük veriyi işlemek için yetersiz kalmaktadır. Öncelikle yapısal olmayan verilerin yapılandırılması ve anlamlı bir hale getirilmesi gerekir. Buna en güzel örneği olan Google, devasa büyüklükteki verileri kendi geliştirdiği teknolojilerle saklayıp analizini yapmaktadır. Verileri tutma işini Google File System [37], analiz etme kısmını MapReduce [38] ve saklama kısmını Bigtable [39] yapmaktadır.

Veriden anlamlı bilgi çıkarma çalışmaları, ticari amaçlar, reklam, pazarlama stratejileri geliştirme, güvenlik uygulamaları gibi farklı alanlarda büyük önem kazanmıştır. Firmalar analizler sonucunda çıkan anlamlı verilerden müşteriye özel kampanyalar veya tanıtımlar düzenleyip satış oranlarını arttırmaya çalışmaktadırlar. Güvenlik kuruluşları insanların veya uygulamaların web üzerinde bıraktığı izleri (loglar) kullanarak tehditleri önceden anlamaya ve önlem almaya çalışırlar. Web sunucu logları gibi çok büyük dosyaları işlemek için Büyük veri teknolojileri kullanılabilir.

Büyük verinin başlangıçta üç önemli özelliği olduğu kabul edilmiştir. Bunlar 3V olarak adlandırılan Volume (Hacim), Variety (Çeşitlilik) ve Velocity (Hız) özellikleridir. Daha sonra bu özelliklere Value (Değer) kavramı da eklenmiştir. Bu kavramlar kısaca şu şekilde açıklanabilir:

2.4.1. Hacim (Volume)

Zaman geçtikçe verinin de büyüklüğü hızla artmaktadır. Eskiden depolama için gereken maliyet yüksek olduğu için verinin depolanması sorun oluştururdu oysa bugün depolama maliyetinin düşmesi ile beraber Google, Facebook gibi firmalar depolanan veriyi hiç bir zaman silinmeyecek şekilde kalıcı olarak depolamaktadırlar. Dolayısıyla bugün büyük hacimlere ulaşan veriyle ilgili tek problem verinin uygun bir şekilde depolanması değil aynı zamanda sorgulanabilmesi ve analizi için gerekli teknolojilerin geliştirilmesidir.

2.4.2. Çeşitlilik (Variety)

Günümüzde elde edilen verilerin çoğunluğu yapısal olmayan şekildedir. Birçok farklı ortamdan elde edilen veri formatları vardır. İçerik ve biçim olarak farklı verilerin işlenmesi önemli bir problemdir.

2.4.3. Hız (Velocity)

Veri üreten kaynakların (sensörler, IOT, sosyal medya, internet, araçlar, mobil cihazlar vb) artması ile beraber veri üretim hızı da artmaktadır. Büyük veri sistemlerinin bu hıza cevap verecek şekilde tasarlanması gerekir.

2.4.4. Değer (Value)

Büyük verinin işlenmesi ve analizi sonucunda ortaya çıkan bilginin, ham veriye göre çok daha değerli olduğunu ifade eder..

Veriden anlamlı bilgi çıkarmak için veri madenciliği ve makine öğrenmesi algoritmaları kullanılır. Ancak verinin boyutu arttıkça analiz için geçen zaman ve maliyet de artmaktadır. Dağıtık sistemler bu durumda devreye girer ve büyük verinin analizinde dağıtık veri madenciliği kullanılır.

2.5. Büyük Veri Teknolojileri

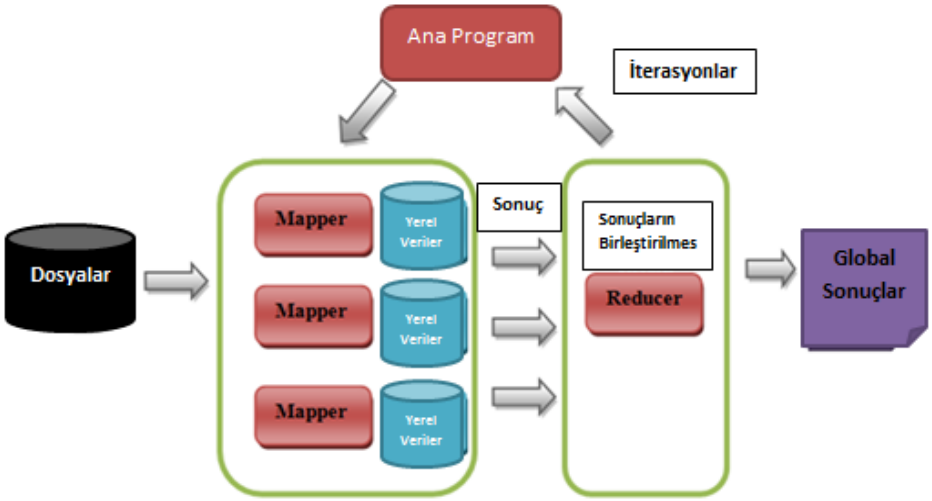
2.5.1. MapReduce

Büyük veriler üzerinde çalışma, bilimsel amaçlar için yapılabileceği gibi kar amaçlı şirketlerin de karar alma, öneri ve tahminlerde bulunma gibi önemli konularda ihtiyaç duydukları bir alandır. Verilerin exponansiyel olarak artışıyla birlikte ortaya çıkan

problemlerle ilk olarak Google, Yahoo, Amazon ve Microsoft gibi ileri teknoloji şirketleri karşı karşıya kalmıştır. Bu şirketler internette yayınlanmakta olan tüm web sitelerini indekslemek, web sitelerini daha popüler yapmak, hangi kitabın daha çok talep edildiğini öğrenmek, hangi kişilere hangi reklamların gösterileceğine karar vermek gibi ihtiyaçlarına çözüm bulabilmek için petabaytlarca veriyi işleyebilecek sistem ve araçlara ihtiyaç duymuşlardır.

İlk olarak Google şirketi (Google Research) yayınladığı makale ile kendi içerisinde veri işleme ile ilgili ihtiyaçları için geliştirdiği MapReduce sistemini duyurmuştur. MapReduce modelinde Google'ın GFS (Google File System) [37] adını verdiği dağıtık dosya sistemi üzerinde sakladığı petabaytlarca veriyi mapperlara <key>,<value> çiftleri halinde giriş olarak alıp intermediate value adı verilen ara veri formatında reducerlara gönderir, reducerlar da intermediate <key>,<value> olarak aldıkları giriş verileri üzerinde gruplama yaparak çıkış dosyalarına yazar. Bu yüksek performanslı dağıtık veri işleme yönteminde master düğümü verilerin dağıtılması, hata ihtimaline karşı yedeklenmesi gibi tüm yönetimsel işleri yürüten birimdir [38].

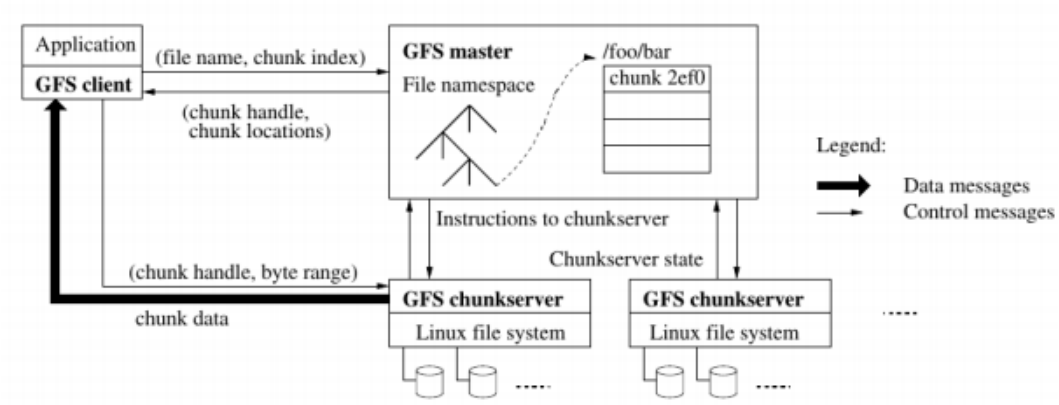
MapReduce programlama büyük miktarda veri (metin, ses, görüntü vb) kullanılması gereken çalışmalarda kullanılmaktadır. Örneğin elimizde yüksek miktarda görüntü dosyası olsun ve bu görüntüler üzerinde yüz çıkartımı işlemi yapmak isteyelim. MapReduce programlama tekniği böyle bir uygulama için oldukça elverişlidir [40]. MapReduce ile görüntü analizi [41], doğal dil işleme [42], büyük boyutlu matris hesaplamaları [43], biyoinformatik araçlarının geliştirilmesi [44] gibi birçok alanda yapılmış çalışmalar mevcuttur.



Şekil 2.3. MapReduce Adımları

2.5.2. Google Dosya Sistemi

Google File System (GFS) [37], Google' ın kendine özel olarak geliştirdiği geniş ölçekli, dağıtık loglanabilir, kontrol altında tutulabilen bir dosya sistemidir.



Şekil 2.4. Google File System mimarisi [45]

GFS mimarisinde bir adet master sunucu ve birden çok yığın sunucuları yer alır. Master sunucuda metaveri bulunur. Veriler yığın sunucularda tutulur. Master sunucuyla yığın sunucuların haberleşmesi kalp atışı adı verilen mesajlarla sağlanır. Masterın görevleri isim alanının yönetilmesi, yığınların yaratılması, kopyalanması, dengelenmesi ve eski yığınların silinmesidir [46].

2.5.3. Hadoop

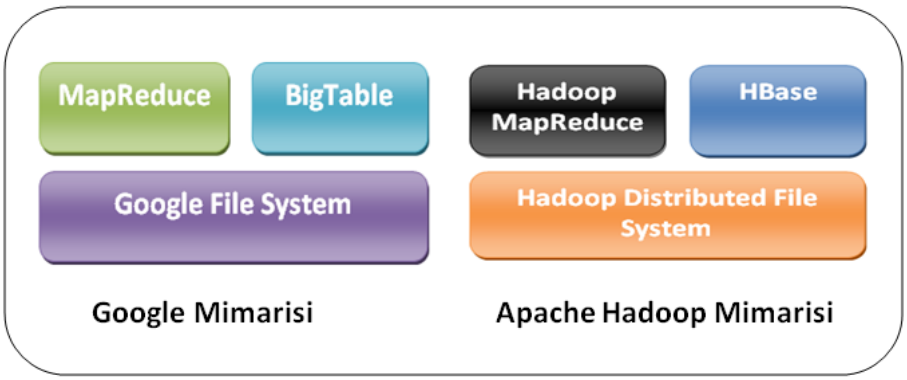
Hadoop Apache tarafından geliştirilmiş, MapReduce programlarının yazılması için gerekli ortamı sağlayan ve dağıtık dosya sistemini oluşturan açık kaynaklı bir kütüphanedir [47]. MapReduce platformu genel amaçlı bilgisayarlardan oluşan bir kümeyle oluşturulabileceği gibi Bulut Sistemi üzerindeki sanal makinelerle de oluşturulmaktadır [48], [49].

Hadoop, tek bir sunucu üzerinde single-node olarak çalıştırılabileceği gibi, yüzlerce hatta binlerce sunucudan oluşan sunucu kümeler (cluster) üzerinde multi-node olarak da çalıştırılabilir.

Hadoop iki ana bileşenden oluşmaktadır: HDFS ve 2.5.1. bölümünde anlatılan MapReduce yaklaşımının implementasyonu.

2.5.3.1. Hadoop Dağıtık Dosya Sistemi (Hadoop Distibuted File System-HDFS)

HDFS [50], Google Dosya Sistemi (Google File System - GFS) temel alınarak oluşturulmuş hadoop içindeki dağıtık dosya sistemidir. Verinin 64MB veya 128MB bloklar halinde saklanmasını sağlar. HDFS, sunucu disklerini bir araya getirdiği için çok büyük miktarda verinin dağıtık olarak depolanmasına olanak sağlar. Veriler bloklara ayrılır ve her bir blok küme üzerindeki farklı disklere dağıtılır. Varsayılan değer olarak her bir veri bloğunun üç farklı kopyası, üç farklı sunucuda tutulur. Böylece herhangi bir disk veya sunucu arızası durumunda veri kaybının önüne geçilmiş olur.



Şekil 2.5. Google – Apache Hadoop Mimarisi

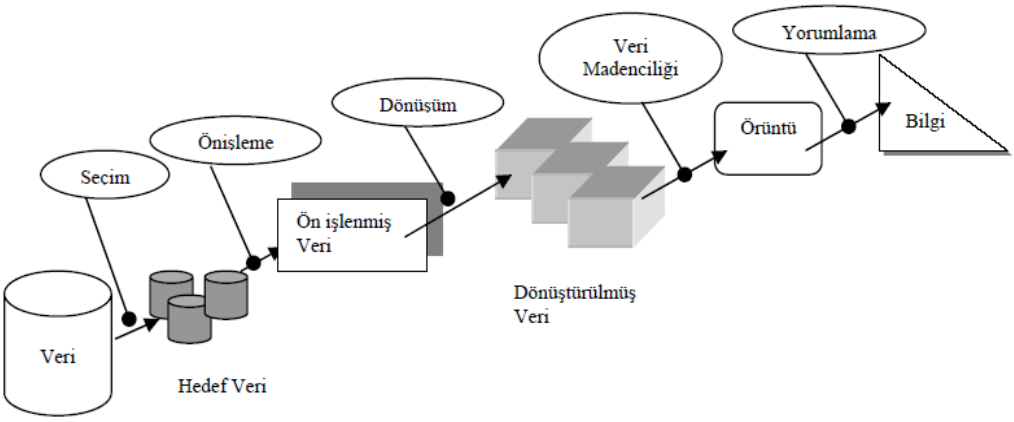
Şekil 2.6. HDFS'te blokların düğümlere dağılımı

3. VERİ MADENCİLİĞİ

3.1. Klasik Veri Madenciliği

Bilişim dünyasındaki gelişmelerle beraber depolanan verinin miktarı da artmıştır. İnternetin yaygınlaşması veri artış hızına büyük bir ivme katmış ve veri miktarını devasa boyutlara ulaştırmıştır. Bunun sonucu oalrak da bu verilerden anlamlı bilgi çıkarmak için kullanılan teknikler çok önem kazanmıştır. Veri madenciliği, depolanan veri setlerinde saklı durumdaki anlamlı örüntüleri bulma işlemidir. Yani genel olarak veri madenciliği veri tabanlarından, doğru, anlaşılır ve potansiyel olarak kullanışlı bilginin çıkarılmasıdır.

Veri madenciliği veri tabanlarından bilgi keşfi (VTBK) işleminin temel bileşenlerinden birisini oluşturmakla beraber VTBK veri madenciliği dışında içerisinde farklı süreçleri barındırmaktadır. Şekil 1'de veri madenciliğinde veriyi anlamlı bilgiye dönüştürme aşamaları gösterilmektedir.



Şekil 3.1. Veri tabanından bilgi keşfi aşamaları [51]

VTBK sürecini oluşturan adımlar aşağıdaki şekilde sıralanmaktadır [52]:

• Veri temizleme: Veri setindeki verilerin hatalı veya tutarsız olması sonucu gürültü meydana gelir. Gürültünün temizlenmesi için verilerdeki eksik bilgi içeren kayıtlar dahil edilmeyebilir, eksik olan değerler yerine sabit bir değer kullanılabilir, kayıp veriler yerine diğer değerlerin ortalaması alınarak kayıp veriler yerine ortalama değer kullanılabilir.

• Veri bütünleştirme: Farklı veri setlerindeki verilerin birlikte kullanılabilmesi için tek türe dönüştürme işlemidir.

• Veri seçme: Veri madenciliği uygulamalarındaki sonucu etkilemeyen verilerin sayısı azaltılabilir.

• Veri dönüşümü: Veriyi kullanılacak veri madenciliği tekniğindeki hali için içeriğini koruyacak şekilde dönüşümünü gerçekleştirmek.

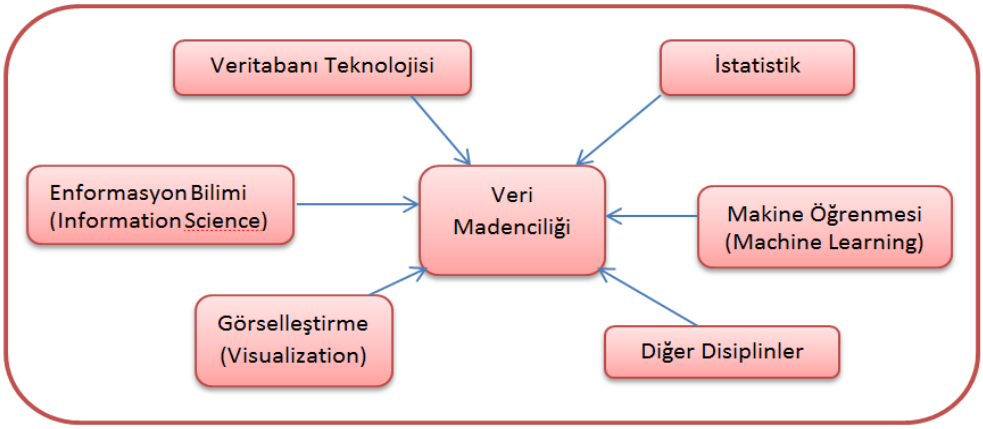
• Veri madenciliği: Veri madenciliği ile ilgili algoritmaları kullanma işlemidir.

• Örüntü değerlendirme: Veri örüntülerini yakaladıktan sonra değerlendirme yapma işlemidir.

• Bilgi sunumu: Veri madenciliği algoritmalarından sonra elde edilen anlamlı bilginin kullanıcıya sunumudur.

Veri setlerinde anlamlı bilgi çıkarılma aşamasında ilk olarak gerekli veriler alınır, sonra alınan veriler üzerinde sınıflandırma gerçekleşir ve bu sınıflandırma sonucunda veriler işlenir [53]. Sınıflandırma aşamasında genetik algoritmalar, karar ağaçları gibi teknikler kullanılır.

Veri madenciliği Şekil 2'de görüldüğü gibi veri tabanı sistemleri, istatistik, makine öğrenmesi, görselleştirme ve enformasyon bilimini içeren disiplinler arası bir alandır [54].



Şekil 3.2. Veri Madenciliğini Oluşturan Disiplinler

3.1.1. Veri Madenciliği Teknikleri

Veri madenciliği teknikleri 3 ana başlıkta incelenir:

• Sınıflama (Classification),

• Kümeleme (Clustering),
• Birliktelik kuralları ve sıralı örüntüler (Association rules and sequential patterns).

3.1.1.1. Sınıflama

Sınıflama, veri madenciliği teknikleri arasında en çok kullanılan yöntemdir. Yeni durumun özelliklerine göre mevcut durumlardan hangisine ait olduğunu belirleme işlemine sınıflama denir.

Var olan verilere göre sınıflar oluşturulduğundan denetimli öğrenme olarak da ifade edilir. Sınıflandırma, sınıfsal değerler ile ilgilenir.

Temel sınıflama algoritmaları aşağıdadır:

- Genetik Algoritma,
- Karar Ağaçları,
- Naive Bayes,
- Sinir ağları,
- Çoklu Regresyon,
- K-En Yakın Komşu.

Veri madenciliğinin en çok kullandığı teknik karar ağaçlarıdır. Karar ağaçlar diğer algoritmalara göre kolay eğitilebilir, daha hızlı sonuçlar verir, her adımı kolaylıkla izlenip anlaşılabilirdir. Karar ağaçları böl ve yönet kuralına göre çalışırlar.

3.1.1.2. Kümeleme

Kümeleme, veri setlerindeki bilgileri belirli benzer ölçütlere göre gruplara ayırma işlemidir. Amaç n tane elemanı olan veri setini farklı ölçütlere sahip k tane veri kümesine bölmektir.

Kümeleme bilgiye daha hızlı bir şekilde ulaşmamızı sağlayan, denetimsiz öğrenme kategorisine giren bir yöntemdir. Kümelemede amaç verileri alt kümelere ayırmaktır. Alt kümelere ayrılmak için keşfedilen kurallar yardımıyla bir kaydın hangi alt kümeye girdiği kümeleme yöntemi kullanarak bulunur [54].

Temel kümeleme algoritmaları aşağıdadır:

- K-Means yöntemi,
- K-Median yöntemi,
- Hiyerarşik kümeleme yöntemleri,
- Yoğunluğa dayalı kümeleme yöntemleri.

Kümeleme tekniklerinden en çok kullanılan yöntem K-Means yöntemidir.

3.1.1.3. K-Means Kümeleme Algoritması

K-Means Algoritması 1957 yılında Cox tarafından ortaya atılmış, 1967 yılında J.B. MacQueen tarafından geliştirilmiş en eski kümeleme algoritmasıdır [55]. Gözetimsiz öğrenme yöntemleri arasındadır. K-Means'in atama mekanizması her bir verinin yalnız bir kümeye ait olabilmesini sağlar [54].

K-means kümeleme algoritması veri madenciliğinde veri ön hazırlama ve veri setindeki gizli örüntülerin tanımlanması için en çok kullanılan kümeleme tekniklerinden biridir. K-Means algoritması d boyutlu metrik uzayda verilen n adet nesnenin k adet kümeye bölünmesinin yapılmasıdır. Öncelikle giriş parametresi olarak *k* değerinin verilmesi gerekir. Küme içi benzerliğin yüksek lakin kümeler arası benzerliğin düşük olmasına hedeflenir. Kümelerdeki benzerlik o kümedeki elemanların ortalama değeri ile hesaplanmaktadır, bu da kümenin ağırlık merkezi sayılır [54].

Bu algoritma şu parametreleri alır:

k: kaç küme olacak

d: kaç nesne olacak

Bu nesneler benzersizliklerine göre kümeleme yapılıp geri verilir. Bu algoritmada kümeler arasındaki benzerlik düşük olur.

Algoritma önce k parametresini yani küme sayısını belirler. Hesaplanan k parametresi başlangıçta kümenin ortalamasını (merkezini) temsil etmektedir. Veri setinde bulunan her bir nesne kendisi ile küme ortalaması arasındaki uzaklığa göre kendisine en çok benzeyen

kümeye atanır ve tekrar her bir küme için küme ortalamaları hesaplanır. Kriter fonksiyonu ortak bir noktada birlesene kadar bu iterasyon devam eder. K-Means kümeleme metodunun değerlendirilmesinde en çok karesel hata kriteri SSE kullanılır. Kümeleme en iyi sonucu vermesi için SSE değerinin çok az olması gerekir. Noktaların oldukları demetin merkez noktalarına olan uzaklıklarının karelerinin toplamı aşağıdaki gibi hesaplanmaktadır [56], [57].

$$\text{SSE}= \sum_{i=1}^{k} \sum_{x \in c_i} dist^2(m_i, x)$$

x: C_i kümesinde bulunan bir nokta,

m_i: C_i kümesinin merkez noktası

Başlangıç küme merkezlerinin seçimi K-Means'in sonucunu önemli oranda etkiler. Başlangıç noktalarının belirlenmesinde farklı yöntemler vardır. Bu yöntemlerin bazıları aşağıdaki gibidir [58]:

• *k* sayısı kadar rastgele veri seçilip küme merkezleri olarak atanır

•Veriler rastgele *k* tane kümeye atanır ve küme ortalamaları alınarak başlangıç küme merkezleri belirlenir

• En uç değerlere sahip veriler küme merkezleri olarak seçilir.

• Veri setinin merkezine en yakın noktalar başlangıç noktaları olarak seçilir.

K-means algoritması aşağıdaki şekilde özetlenebilir [59]:

Girdi: n nesneden oluşan veritabanları ve k küme sayısı

Çıktı: Hata kareler toplamını minimum yapan k küme seti

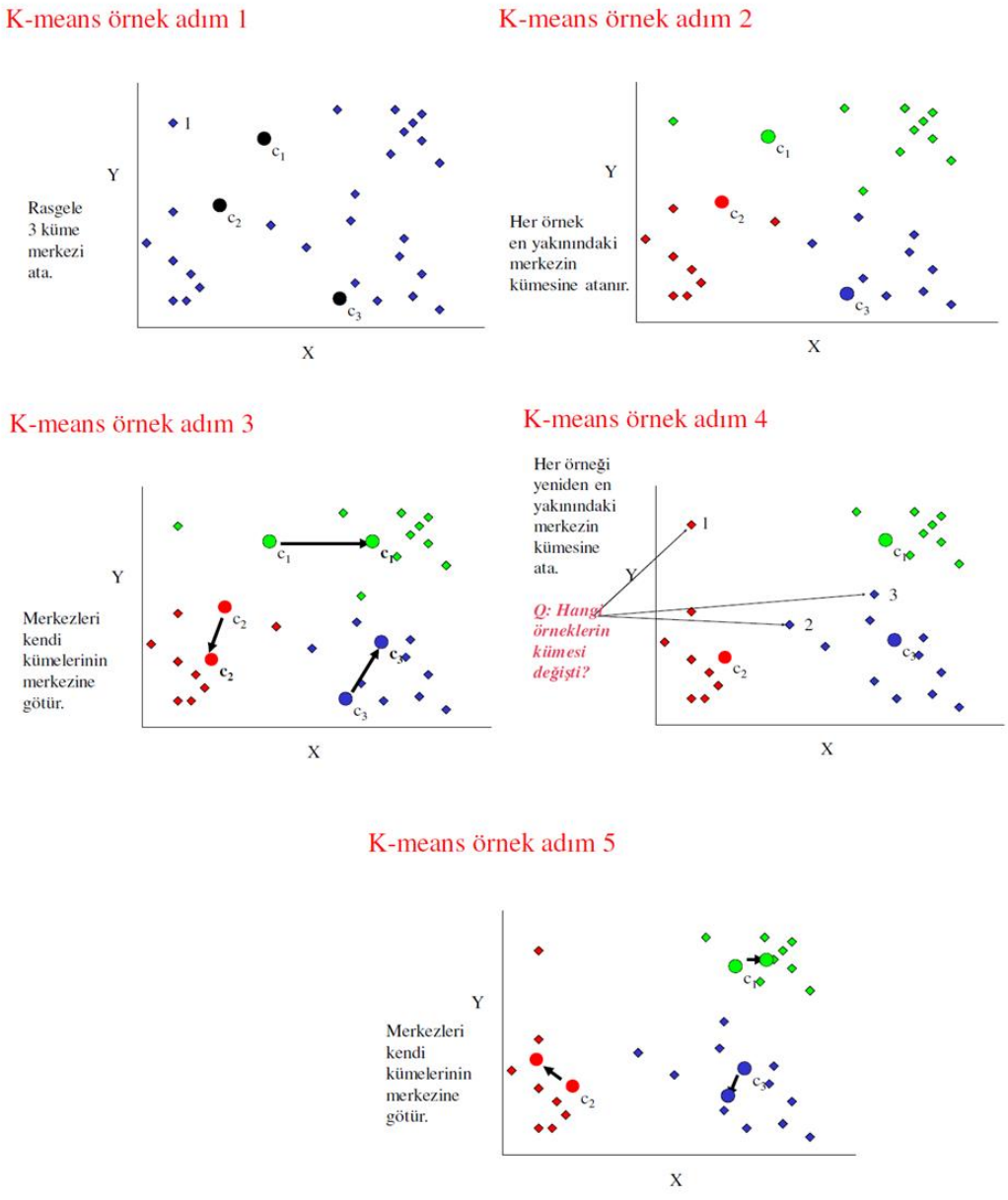
Adım 1: İlk küme merkezleri belirlenir.

Adım 2: Her nesnenin seçilen merkez noktalara olan uzaklığı hesaplanır. Elde edilen sonuçlara göre tüm nesneler k adet kümeden kendilerine en yakın olan kümeye yerleştirilir.

Adım 3: Oluşan kümelerin yeni merkez noktaları o kümedeki tüm nesnelerin ortalama değeri ile değiştirilir.

Adım 4: Merkez noktalar değişmeyene kadar Adım 2 ve Adım 3 tekrarlanır.

Örnek olarak K- means algoritmasının adım adım uygulanışı [60]:



Şekil 3.3. K-Means Kümeleme Örnek [27]

Örnek olarak 4 adet objenin olduğu ve her bir objenin iki özelliğe sahip olduğunu varsayalım.

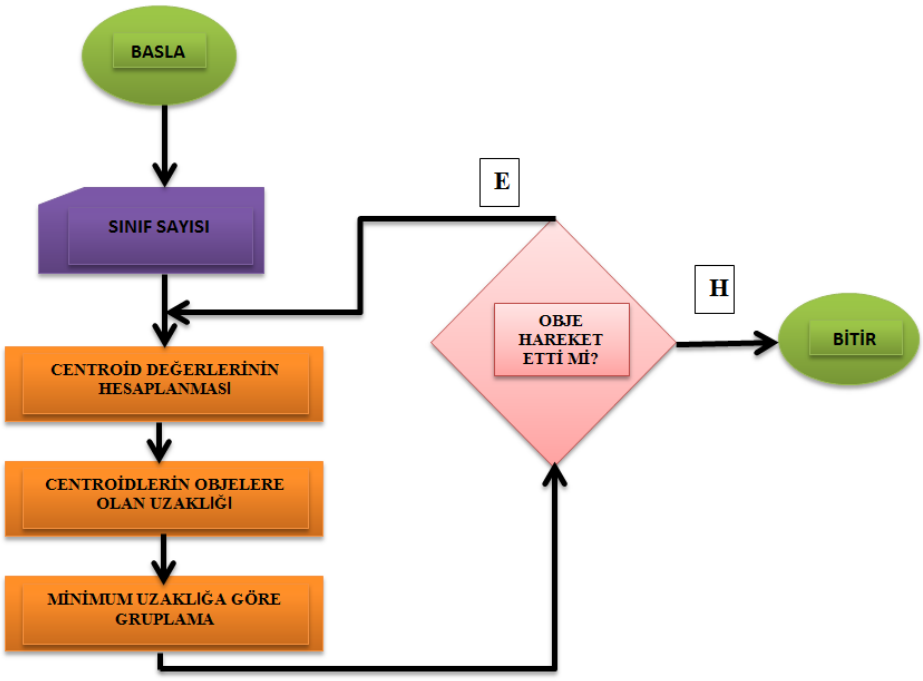
Tablo 3.1. A, B, C ve D nesnelerinin özellikleri

Nesne	Özellik 1	Özellik 2
A	**1**	**1**
B	**2**	**1**
C	**4**	**3**
D	**5**	**4**

Öncelikle bilinmesi gereken objelerin kaç kümeye ayrılacağıdır Bu örnek için küme C1 ve C2 adında iki kümemiz olsun. Başlangıç centroid'leri olarak veri setinden rasgele k nokta seçilebilir.

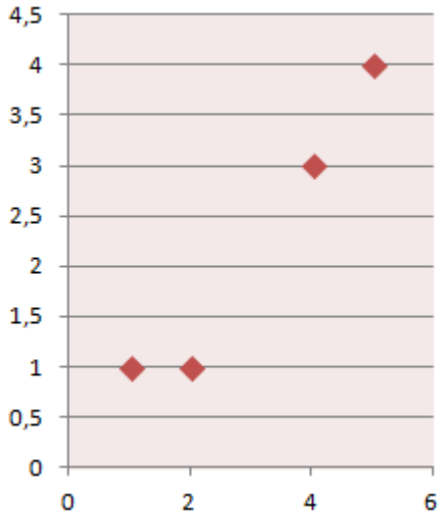
Daha sonra K-means algoritmasında, cluster'lar içerisinde yer alan objeler hareketsiz kalıncaya kadar yani yer degistirmeyinceye kadar üç aşamadan olusan işlem tekrarlanır.

- Centroid noktalarına karar verilir.
- Her objenin centroid noktalarına olan uzaklıkları hesaplanır.
- Her obje minimum uzaklıgı sahip oldugu cluster'a atanır.



Şekil 3.4. K-Means Algoritması Adımları

Her bir objeyi özellik uzayında (X,Y) olarak gösterecek olursak:



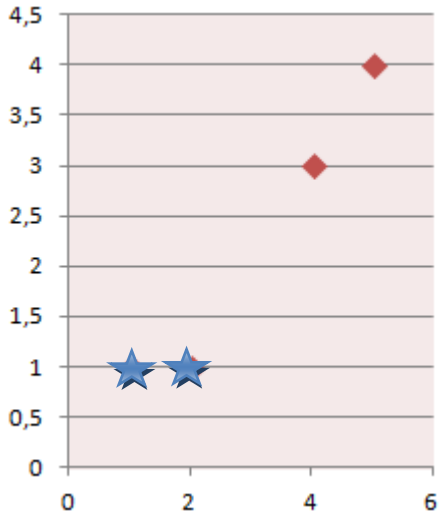
Şekil 3.5. A, B, C ve D nesnelerinin X-Y düzlemindeki ilk konumları

İterasyon 0:

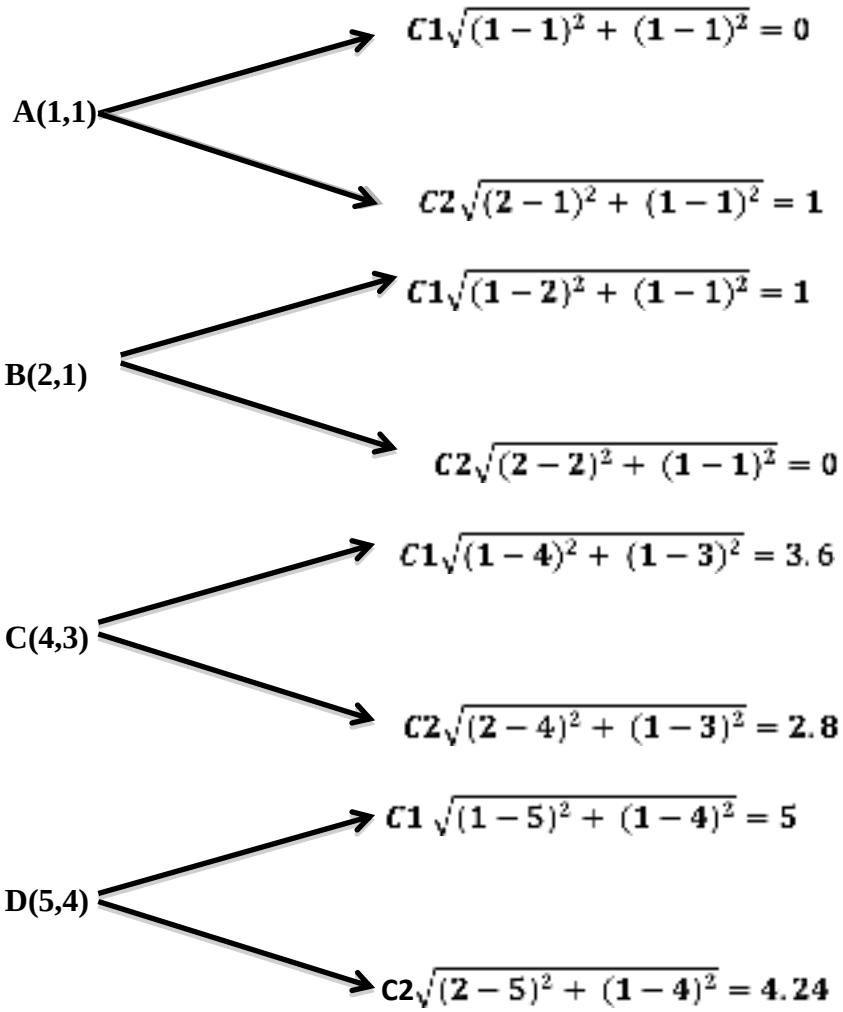
Adım 1: Başlangıç centroid değerleri:

İlk centroid degerleri olarak A ve B 'yi alalım.

Centroid koordinatları C1=(1,1) ve C2=(2,1) olsun.



Şekil 3.6. Merkezleri A ve B olan C1 ve C2 küme konumları



Adım 2: Objelerin centroid'lere olan mesafesinin ölçülmesi :

Her bir obje ile cluster centroid'i arasındaki mesafeyi ölçmek için Öklit Uzaklığı kullanılır.

Sıfırıncı iterasyonda elde edilen distance matrisi

$$D^0 = \begin{vmatrix} 0 & 1 & 3.6 & 5 \\ 1 & 0 & 2.8 & 4.24 \end{vmatrix}$$

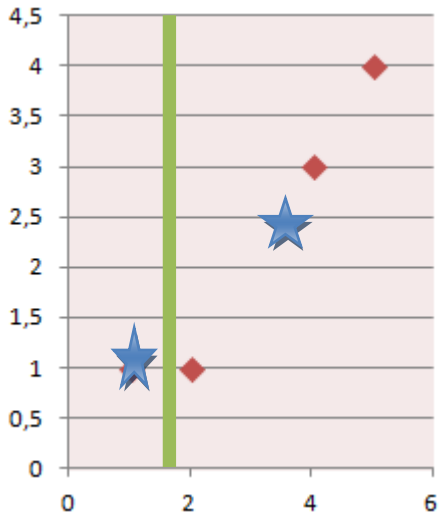
Adım 3: Her bir objenin minimum değeri hangi cluster'a ait ise, o obje artık o cluster'a ait demektir.

$$G^0 = \begin{matrix} A & B & C & D \\ \end{matrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{matrix} \text{CLUSTER-1} \\ \text{CLUSTER-2} \end{matrix}$$

İterasyon 1:

Adım 1: Her bir cluster için yeni centroid değerleri hesaplanır. Her cluster içerisinde yer olan objelerin ortalama değerleri alınır.

Birinci cluster içerisinde sadece A (1,1) olduğundan Cluster-1 'in yeni centroid değerleri C1 (1,1)' dir. Cluster-2 içerisinde B-C-D olduğundan



Şekil 3.7. C1 ve C2 küme konumları

$$C2 = \left(\frac{2+4+5}{3}, \frac{1+3+4}{3}\right) = (3.67, 2.67)$$

Adım 2: Tüm objelerin yeni centroid değerlerine olan mesafeleri yeniden hesaplanır.

A(1,1)

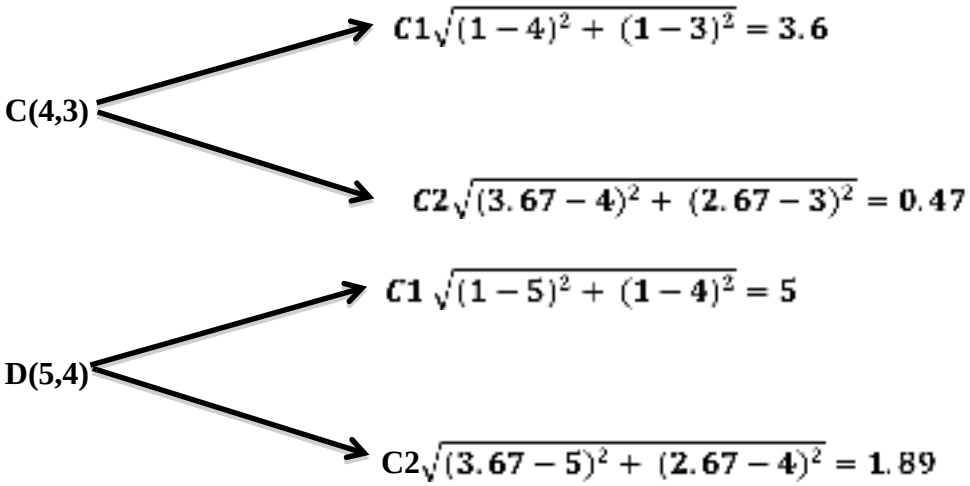
$$C1\sqrt{(1-1)^2 + (1-1)^2} = 0$$

$$C2\sqrt{(3.67-1)^2 + (2.67-1)^2} = 3.14$$

B(2,1)

$$C1\sqrt{(1-2)^2 + (1-1)^2} = 1$$

$$C2\sqrt{(3.67-2)^2 + (2.67-1)^2} = 2.36$$



Birinci iterasyonda elde edilen distance matrisi

$$D^1 = \begin{vmatrix} 0 & 1 & 3.6 & 5 \\ 3.14 & 2.36 & 0.47 & 1.89 \end{vmatrix}$$

Adım 3: Her bir objenin minimum değeri hangi cluster'a ait ise, o obje artık o cluster'a ait demektir.

$$G^1 = \begin{matrix} \\ \text{CLUSTER-1} \\ \text{CLUSTER-2} \end{matrix} \begin{matrix} \begin{matrix} A & B & C & D \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \end{matrix}$$

Objelerin yerleri değiştiği için aynı iterasyona devam edilir.

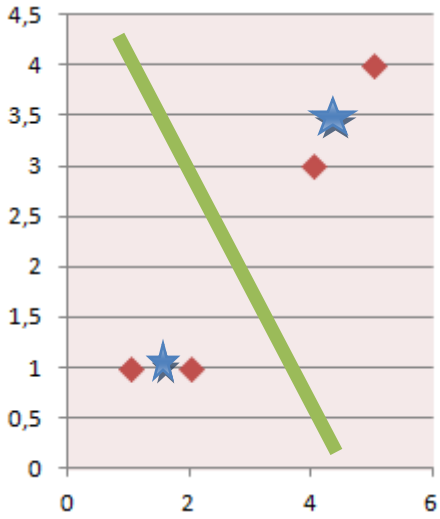
İterasyon 2:

Adım 1: Her bir cluster için yeni centroid değerleri hesaplanır. Her cluster içerisinde yer olan objelerin ortalama değerleri alınır.

Birinci cluster içerisinde sadece A(1,1) ve B(2,1) olduğundan Cluster-1 'in yeni centroid değerleri C1 dir. Cluster-2 içerisinde C(4,3) ve D(5,4) olduğundan:

$$C1 = \left(\frac{1+2}{2}, \frac{1+1}{2}\right) = (1.5, 1)$$

$$C2 = \left(\frac{4+5}{2}, \frac{3+4}{2}\right) = (4.5, 3.5)$$



Şekil 3.8. A, B, C ve D nesnelerinin X-Y düzlemindeki son konumları

Adım 2: Tüm objelerin yeni centroid değerlerine olan mesafeleri yeniden hesaplanır.

A(1,1)

$$C1\sqrt{(1.5-1)^2+(1-1)^2}=0.5$$

$$C2\sqrt{(4.5-1)^2+(3.5-1)^2}=4.3$$

B(2,1)

$$C1\sqrt{(1.5-2)^2+(1-1)^2}=0.5$$

$$C2\sqrt{(4.5-2)^2+(3.5-1)^2}=3.54$$

C(4,3)

$$C1\sqrt{(1.5-4)^2+(1-3)^2}=3.2$$

$$C2\sqrt{(4.5-4)^2+(3.5-3)^2}=0.71$$

D(5,4)

$$C1\sqrt{(1.5-5)^2+(1-4)^2}=4.61$$

$$C2\sqrt{(4.5-5)^2+(3.5-4)^2}=0.71$$

İkinci iterasyonda elde edilen distance matrisi

$$D^2 = \begin{vmatrix} 0.5 & 0.5 & 3.2 & 4.61 \\ 4.3 & 3.54 & 0.71 & 0.71 \end{vmatrix}$$

Adım 3: Herbir objenin minimum değeri hangi cluster'a ait ise, o obje artık o cluster'a ait demektir.

$$G^2 = \begin{matrix} \\ \text{CLUSTER-1} \\ \text{CLUSTER-2} \end{matrix} \begin{matrix} \begin{matrix} A & B & C & D \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \end{matrix}$$

Objelerin yerleri değişmediği için işlem tamamlanmıştır. G2 = G1

Tablo 3.2. Nesne – Cluster Tablosu

Nesne	Özellik 1	Özellik 2
A	**1**	**1**
B	**2**	**1**
C	**4**	**3**
D	**5**	**4**

CLUSTER
1
1
2
2

3.2. Mekansal Veri Madenciliği

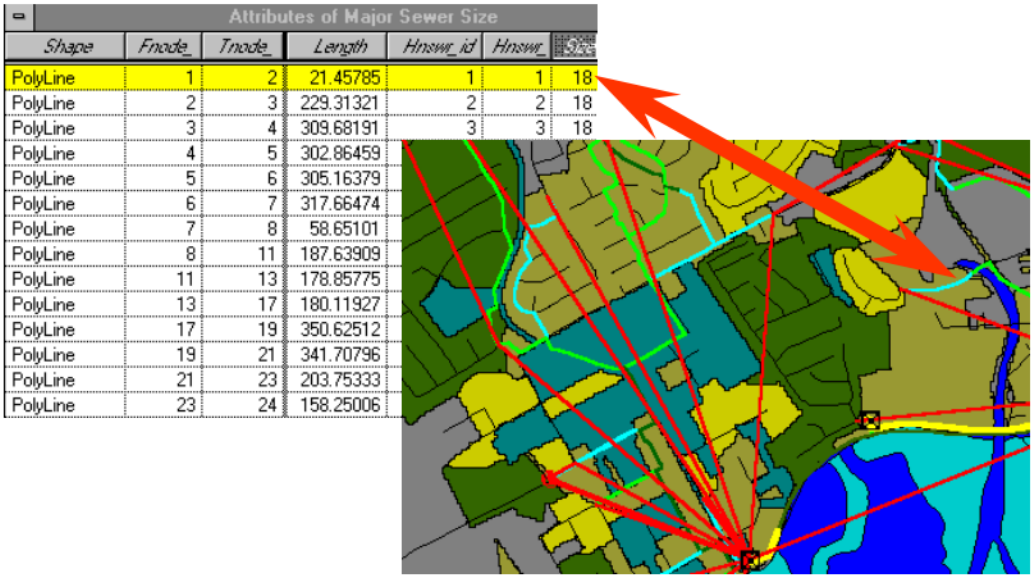
3.2.1. Mekansal Veri

Mekansal veya coğrafi veri, içerisinde bir tür mekânsal referans taşıyan ve dolayısıyla bir veya birden fazla konum bilgisi içeren özel bir veri türüdür. Coğrafi veriler genelde iki kısımdan oluşur: verinin içeriğini anlatan öznitelik verileri (meta-data) ve mekansal veriler (koordinatlar)

Öznitelik verileri bir konumun özelliklerini barındırır. Mekansal veri içerisindeki konum bilgileri ise nokta, çizgi ve poligon yapılarıyla ifade edilir. Örneğin bir binanın koordinatları ve şekli mekânsal veridir, ancak o binanın adresi, kapı numarası, kat adedi vb bilgiler öznitelik verileridir.

Mekansal veriler, oluşturuldukları koordinat referans sistemleri, farklı CBS programları, platform farklılıkları gibi birçok etken sebebiyle format dönüşümüne ihtiyaç duyarlar.

Genel olarak mekânsal veriler depolama ve sunum açısından iki farklı kategoride incelenir: Vektör veri ve Raster veri.



Attributes of Major Sewer Size

Shape	Fnode_	Tnode_	Length	Hnswr_id	Hnswr_	Size
PolyLine	1	2	21.45785	1	1	18
PolyLine	2	3	229.31321	2	2	18
PolyLine	3	4	309.68191	3	3	18
PolyLine	4	5	302.86459			
PolyLine	5	6	305.16379			
PolyLine	6	7	317.66474			
PolyLine	7	8	58.65101			
PolyLine	8	11	187.63909			
PolyLine	11	13	178.85775			
PolyLine	13	17	180.11927			
PolyLine	17	19	350.62512			
PolyLine	19	21	341.70796			
PolyLine	21	23	203.75333			
PolyLine	23	24	158.25006			

Şekil 3.9. Vektör ve Raster Veriler [61]

3.2.1.1. Vektör Veri

Vektör veriler, nokta, çizgi ve poligon türlerinden oluşur.

- Nokta veriler: Elektrik direkleri, duraklar, hastaneler, okullar, spor salonları, alışveriş merkezleri gibi tek bir konumu belirten veriler, haritalar üzerinde tek bir nokta ile ifade edilebilmektedirler.
- Çizgi veriler: Elektrik hatları, telefon hatları, su ve kanalizasyon şebekeleri, yollar, akarsular gibi birçok farklı noktanın birleşmesi ile oluşan verilerdir.
- Poligon veya alan veriler: Şehirler, denizler, göller, ormanlık araziler gibi noktaların birleşmesi ile oluşan, çizgi verilerinden farklı olarak, belirli bir noktadan başlayıp tekrar aynı noktada son bulan verilerdir.

Noktalar, çizgiler ve poligonlara ait mekansal veriler, x ve y koordinatları belirtilerek saklanır. Bir noktanın yeri, tek bir (x,y) koordinatıyla tanımlanabilir. Yollar ve akarsular gibi çizgisel veriler, (x,y) koordinatlarından oluşan noktalar dizisi şeklinde tanımlanır. Şehir veya ülke sınırları gibi poligon verileri ise, (x,y) koordinatlarından oluşan ve başlangıç ve bitiş noktaları aynı olan noktalar dizisi şeklinde saklanabilir.

Vektör verileri, veritabanlarında birbirleriyle ilişkilendirilmiş halde saklanırlar. Herhangi bir vektörün sağında ve solunda bulunan diğer vektör verileri de veri tabanında saklıdır. Vektör verilerini birbiriyle konumsal olarak ilişkilendirme ve birleştirme işlemlerine topoloji denilmektedir. Topoloji, kesişen çizgileri ya da üst üste binmiş noktaları veya poligonları tanıyarak, vektör verilerini analiz etme konusunda Coğrafi Bilgi Sistemi uygulamalarına yardımcı olmaktadır [62].

3.2.1.2. Raster Veriler

Raster modeli, vektör modelinin aksine sürekli özellikteki verileri modellemede tercih edilir. Raster veriler, ızgara biçiminde hücrelerde saklanır [63]. Öncelikle görüntüler küçük parçalara ayrılır. Grid adı verilen ızgara şeklindeki her bir hücrede, bölgeye ait öznitelik verilerinin o hücreye düşen değeri gösterilir. Bu hücrelerin her birinde yalnızca bir değer saklanır. Örneğin; yolları belirten bir raster veri dizininde, hücrenin aldığı 4 değeri o yolun bir otoyol olduğunu gösterebilir. Bir yolu belirten hücre sayısı yolun uzunluğu ile orantılı olabilmektedir. Fakat her bir hücrenin büyüklüğü birkaç metreyi ifade edebileceği gibi, birkaç kilometreyi de ifade edebilmektedir. Buna hücrenin çözünürlüğü denilmektedir. Aynı bölgeyi temsil etmede, çözünürlüğü yüksek olan haritalar düşük olanlara göre daha çok hücreye ihtiyaç duyarlar. Uydu görüntüleri, raster modeli kullanılarak analiz edilebilir.

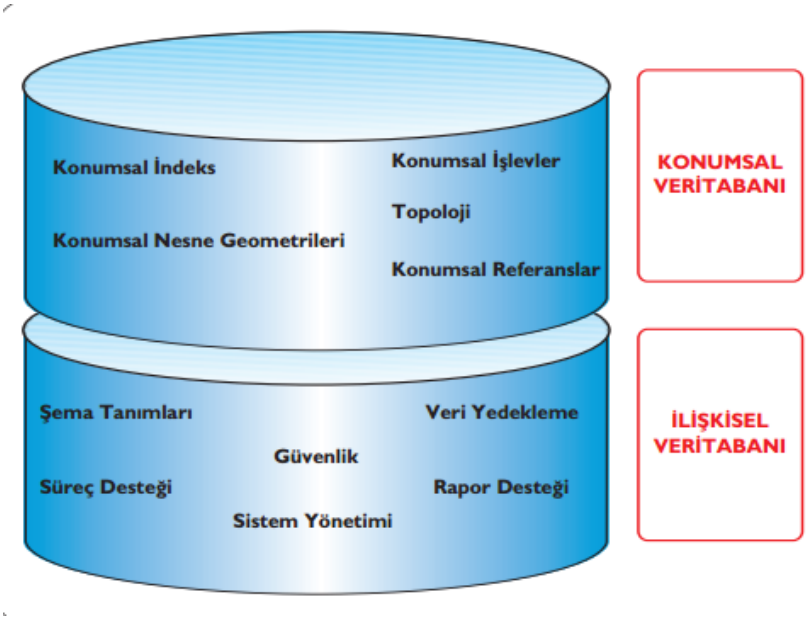
3.2.2. Mekansal Veritabanları

Coğrafi bilgi sistemi uygulamalarının yaygınlaşması ile beraber mekânsal verilerin depolanması ve sorgulanmasını sağlaycak, geleneksel ilişkisel veritabanlarından farklı özellikleri olan veritabanları geliştirilmiştir.

Mekansal veri tabanı, mekansal verilerin saklandığı, işlendiği, sorgulandığı ve bu verilerin yönetildiği veri tabanıdır. Mekansal veri tabanı standart veri tabanı yönetim sistemlerinden farklı olarak, depolanan coğrafi verinin saklanmasını ve sorgulanmasını sağlar. Mekansal veri tabanı kullanımının temel avantajları şunlardır:

- Tüm mekansal verilerin tek merkezde tutulması,
- Veri girişi ve güncelleme işlemlerinin kullanıcı hatalarını en aza indirerek yapılması,
- Konumsal nesnelerin hem topolojik hem de diğer ilişkileriyle birlikte tanımlanması,
- Konumsal sorguları desteklemesi,

- Konumsal nesnelerin geometrik alanlarını veri tabanı bilgi alanında saklaması,
- Kullanıcıların mekansal verilere aynı anda ulaşıp, güncelleme yapabilmesi,
- Basit harita üretimi sağlayacabilmesi.



Şekil 3.10. Mekansal Veritabanı Yapısı [64]

3.2.3. Mekansal Veri Madenciliği (Spatial Data Mining)

Mekansal veri madenciliği kısaca, veri madenciliği metotlarının mekânsal verilere uygulanmasıdır. Burada amaç veri içerisinde mekânsal veya uzaysal örüntüler bulmaktır. Mekansal veriler üzerinde veri madenciliği çalışmalarının geçmişi çok eskiye kadar gitmemektedir. Bunun nedeni de son yıllardaki GPS, sensörler ve benzeri teknolojilerin yaygınlaşmasından önce mekânsal referans taşıyan veri miktarının azlığı ve bu veri türlerinin kısıtlı olmasıdır. Geçmişte, genellikle istatistik yöntemleri kullanılarak mekânsal veriler üzerinde analizler yapılmış ve sonuçlar çıkarılmaya çalışılmıştır.

Ancak son yıllarda bu veriler üzerinde daha nitelikli analizler yapılmaya çalışılmaktadır. Bu analizlerde Veri Madenciliği teknikleri kullanılarak veri içerisinde anlamlı olabilecek örüntüler keşfedilmeye çalışılmaktadır. Bu alanda daha önce yapılan çalışmalarda konum verileri kullanarak önemli yerlerin bulunması, aynı mekanları paylaşan insanların tespiti, trafik yoğunluğu, insan yoğunluğu, aktif mekanların tespit edilmesi gibi çeşitli konular incelenmiştir. Bu çalışmalardan bazı örnekler şunlardır:

Mekansal verilerin analizi birçok farklı sektörde kullanılabilir. Hastalıkların coğrafi dağılımının anlaşılması, arazi kullanımındaki değişimlerin çevreye ve iklim değişikliğine yaptığı etkiler, pazarlama amacıyla lokasyona göre müşteri analizi bunlardan bazılarıdır. Ancak geleneksel veri madenciliği yöntemlerinin mekânsal analizler yapılması amacıyla kullanılması önünde çeşitli engeller bulunmaktadır. Bunlardan bazıları mekânsal veritabanlarının (geospatial databases) genelde çok büyük olmaları, ilişkisel veri yönetimi ile topolojik-mekansal veri yönetiminin farklılaşması ve mekânsal verilerin birçok farklı formatta depolanmasıdır.

4. BULUT BİLİŞİM VE BÜYÜK VERİ TEKNOLOJİLERİ KULLANILARAK MEKANSAL VERİLERİN KÜMELENMESİ

Bu bölümde öncelikle, tezin ana amacı olan büyük mekânsal veri üzerinde veri madenciliği uygulamalarının gerçekleştirilebilmesi için gerekli programların kurulumu anlatılmıştır.

Daha sonra bu programlar ve teknolojiler kullanılarak bir veri madenciliği uygulaması olan kümeleme işleminin mekânsal veri üzerinde uygulanması anlatılmıştır.

Son olarak da büyük mekânsal veri üzerinde kümeleme yapabilmek için küme bilgisayarların kullanılması ve geleneksel yöntemlerle kümelenemeyecek kadar büyük verilerin kümelenmesi ve ilgili performans ölçümleri verilmiştir.

Büyük veri analizi, verinin büyüklüğü ve analizlerin karmaşıklığına göre değişmek üzere, çok sayıda bilgisayarın kullanılmasını gerektirir. Çoğu zaman paralel ve dağıtık hesaplama için, birbiriyle haberleşebilen ve genellikle aynı özelliklere sahip bilgisayarlardan oluşan bilgisayar kümeleri (cluster) kurulur.

Çok sayıda makinanın gerektiği durumlarda, kaynak tasarrufu ve kaynakların verimli kullanılabilmesi için de genelde sanallaştırma çözümleri kullanılır.

Son yıllarda sağladığı esneklik ve çok çeşitli faydaları sebebiyle Bulut Bilişim çözümleri bu sanallaştırma ihtiyacını karşılamak için kullanılmaktadır. Bu tez kapsamında da sunucuların sanallaştırılması ve hesaplama kümelerinin kurulumu için Bulut bilişim çözümleri tercih edilmiştir.

Bulut Bilişim hizmeti, Amazon, Google, Microsoft gibi büyük hizmet sağlayıcılardan kiralanabileceği gibi OpenStack ve benzeri açık kaynaklı çözümlerin kurulumu gerçekleştirerek de sağlanabilir.

Bu tez çalışmasında öncelikle OpenStack kurulumu gerçekleştirilmiş ve bunun üzerinde kurulan Hadoop kümesi ile kümeleme testleri yapılmıştır. Daha sonra Google Cloud üzerinde kurulan Hadoop kümesi üzerinde bu testler gerçekleştirilmiştir.

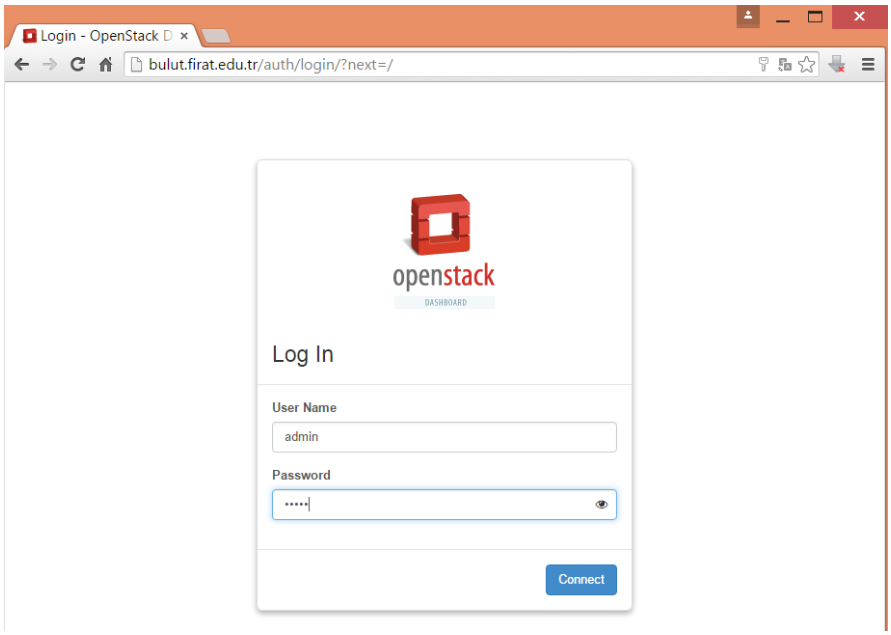
Aşağıda bu çalışmalarda izlenen adımlar, web arayüzleri ve gerçekleştirilen kümeleme çalışmalarının sonuçları verilmiştir.

4.1. OpenStack Yardımıyla Sanal Makinelerin ve Hadoop Kümesinin Ouşturulması

OpenStack açık kaynak kodlu bir Bulut Bilişim Altyapı (IaaS) yazılımıdır. Dünya çapında çok sayıda kullanıcısı vardır ve arkasında önemli şirketlerin desteğini almıştır. Son yıllarda RackSpace gibi büyük ölçekli Bulut hizmet sağlayıcıları tarafından kullanıldığı için de hızlı bir gelişme çizgisi izlemektedir.

Bu tez çalışmasında bulut.firat.edu.tr adresinde çalışan bir sunucuya OpenStack Kilo versiyonu kurulumu gerçekleştirilmiştir. Bu bölümde OpenStack web arayüzü tanıtılmış ve temel bazı işlemlerin nasıl yapıldığı ekran görüntüleri ile gösterilmiştir.

OpenStack yazılımı bu çalışmada Hadoop kümesi için gerekli sanal makineleri oluşturmak için kullanılmış ve toplam 5 sanal makineden oluşan bir küme kurulmuştur.



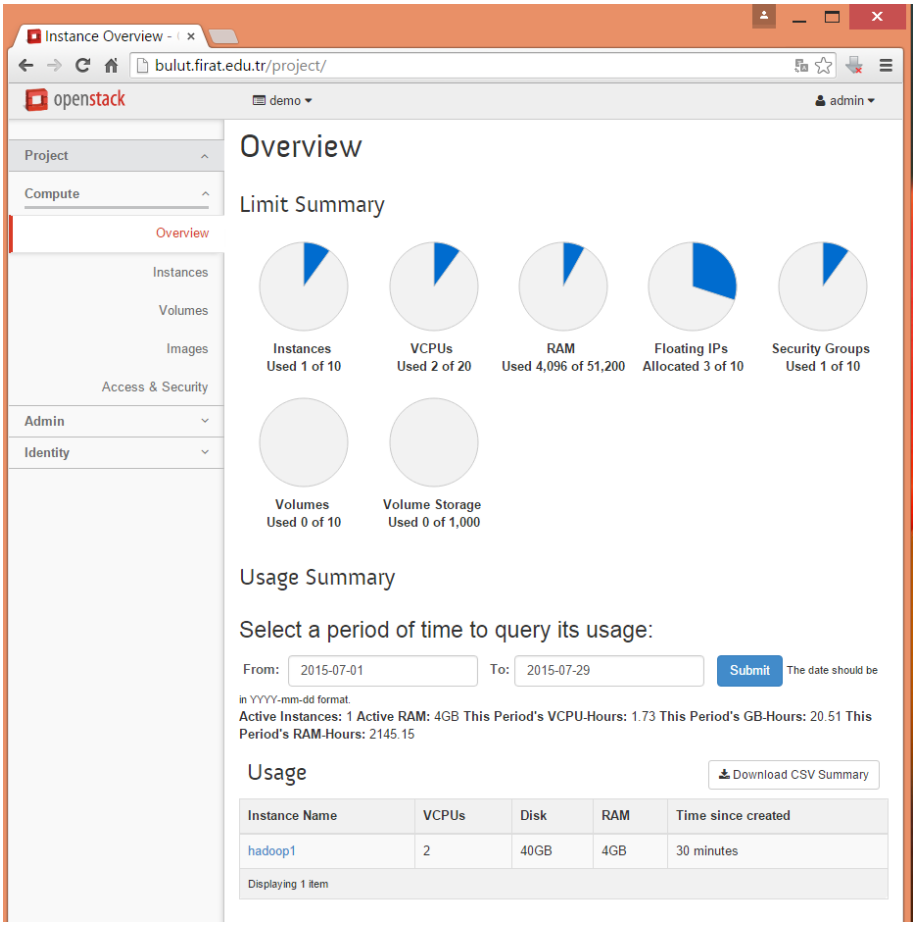
Şekil 4.1. OpenStack giriş ekranı

OpenStack Overview ekranında (Şekil 4.1) sistemin sahip olduğu kaynaklar ve bunların ne kadarının kullanıldığı gösterilmektedir.

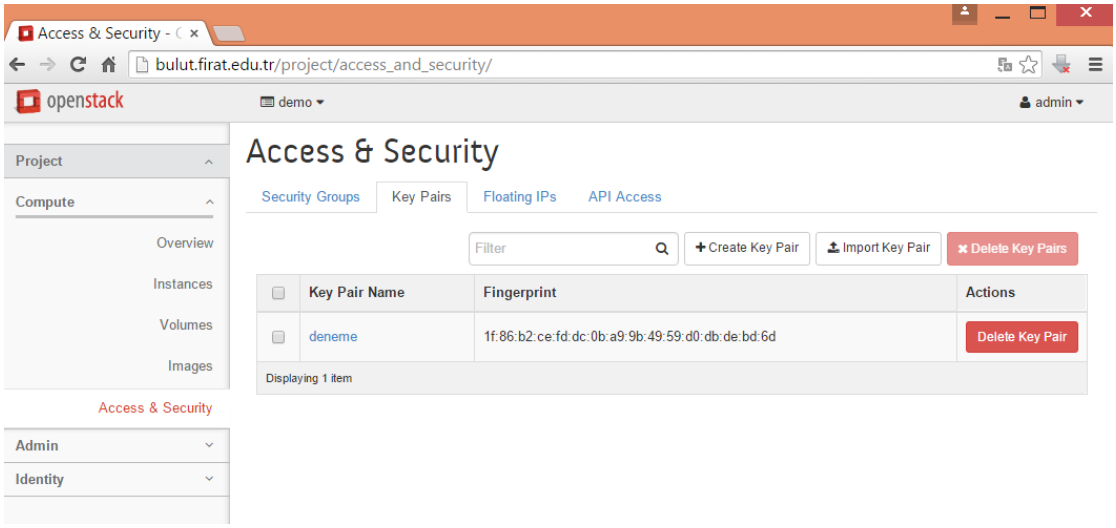
OpenStack çok sayıda kullanıcıya hizmet verebilecek şekilde tasarlanmıştır. Sistem yöneticisi kullanıcı grupları için projeler oluşturabilir ve bu projelere kaynak limitleri (CPU

sayısı, RAM miktarı, depolama miktarı vb) atayabilir. Böylece aynı OpenStack kurulumundan çok sayıda kullanıcı veya kullanıcı grubu birbirlerinden habersiz bir şekilde yararlanabilir. Bu çok-misafirli (multi-tenant) kullanım Bulut sistemlerinin en önemli özelliklerinden birisidir.

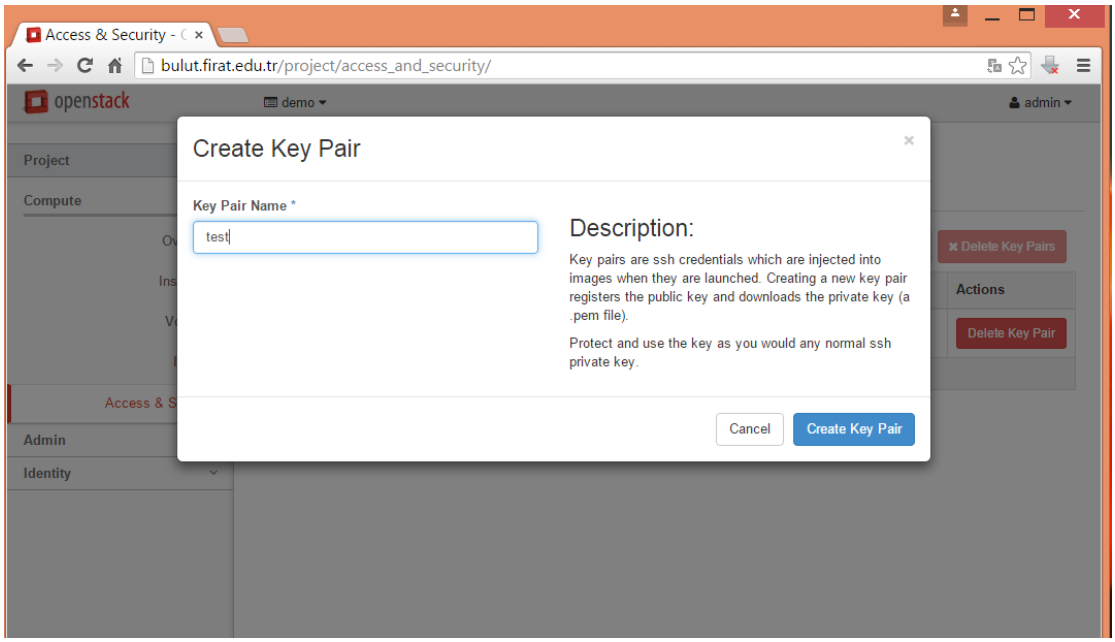
OpenStack üzerinde kurulacak sanal makinelere erişim SSH ile gerçekleştirilebilir. Ancak bu makinelere şifre kullanılarak değil OpenStack tarafından oluşturulacak Key Pair adı verilen özel şifre dosyaları ile erişilebilir. Dolayısıyla ilk yapılması gereken işlerden birisi de proje için bir Key Pair oluşturmaktır (Şekil 4.2). Yeni Key Pair dosyaları oluşturulduktan sonra indirilir ve saklanır.



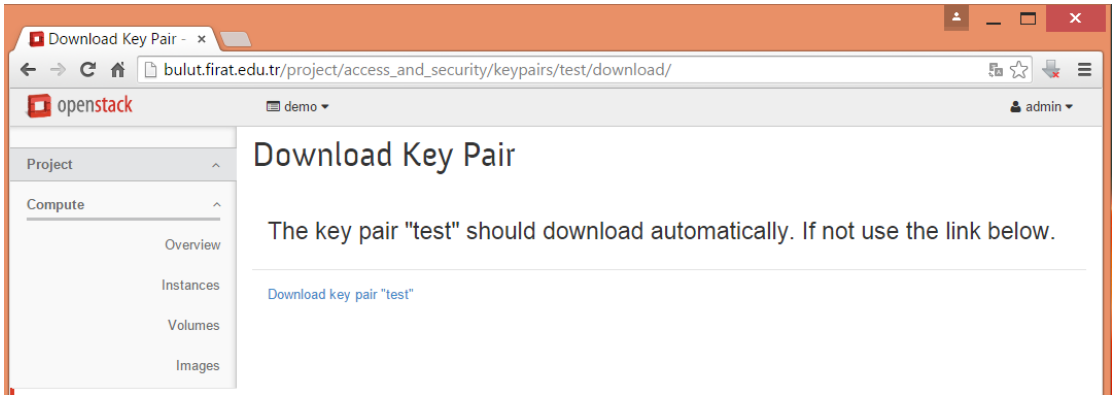
Şekil 4.2. Sistemin genel durumunu gösteren Overview ekranı



Şekil 4.3. Sanal Makinelere erişim için kullanılacak Key Pair listesi



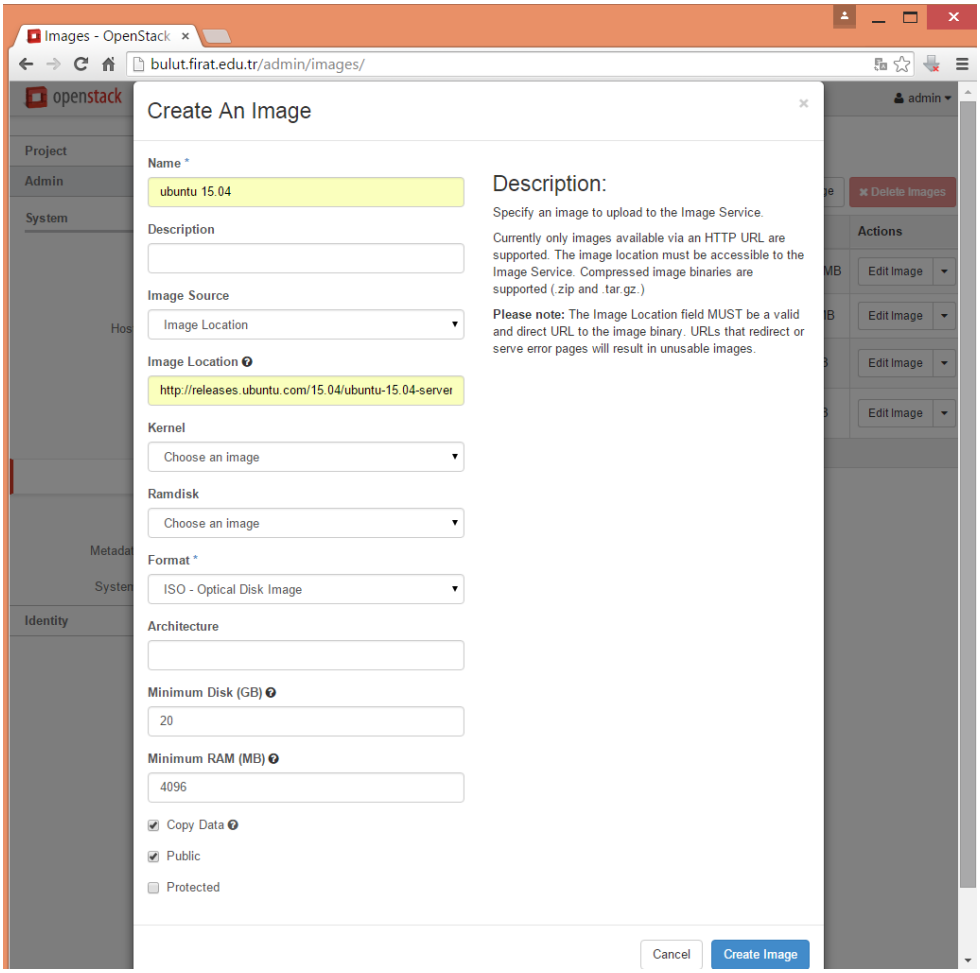
Şekil 4.4. Yeni Key Pair oluşturma ekranı



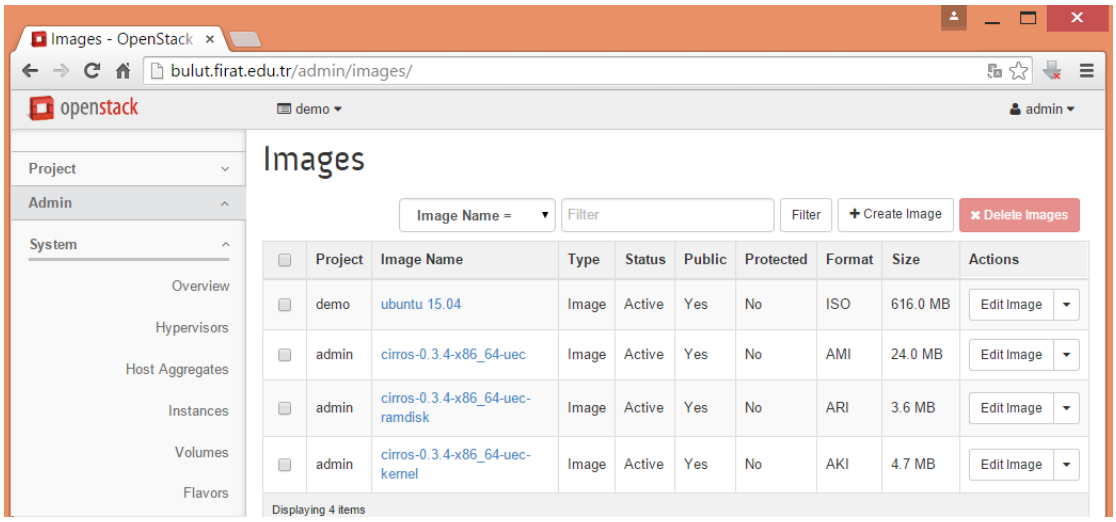
Şekil 4.5. Yeni Key Pair Download ekranı

OpenStack üzerinde oluşturulacak sanal makinelerde çalışacak işletim sistemi imajlarının sisteme eklenmesi gerekmektedir. OpenStack ilk kurulumdan sonra Cirros adı verilen küçük ve basit bir Linux sürümü imajı ile gelmektedir. Ancak bu işletim sistemi Hadoop benzeri sistemleri çalıştıramaz. Dolayısıyla Ubuntu, CentOS veya Windows gibi işletim sistemleri imajlarının eklenmesi gerekir.

Kullanıcı projesinin Images ekranı yardımıyla sisteme yeni imajlar ekleyebilir. İşletim Sisteme imajlar, lokal makineye kopyalanacak ISO dosyaları ile eklenebildiği gibi internette bulunan herhangi bir ISO veya uygun formatlı imajlar da eklenebilmektedir. Örneğin Şekil X4'te internet adresi verilen Ubuntu 15.04 Linux işletim sistemi imajının sisteme eklenmesi gösterilmiştir. OpenStack bu dosyayı internetten indirir ve sisteme ekler.



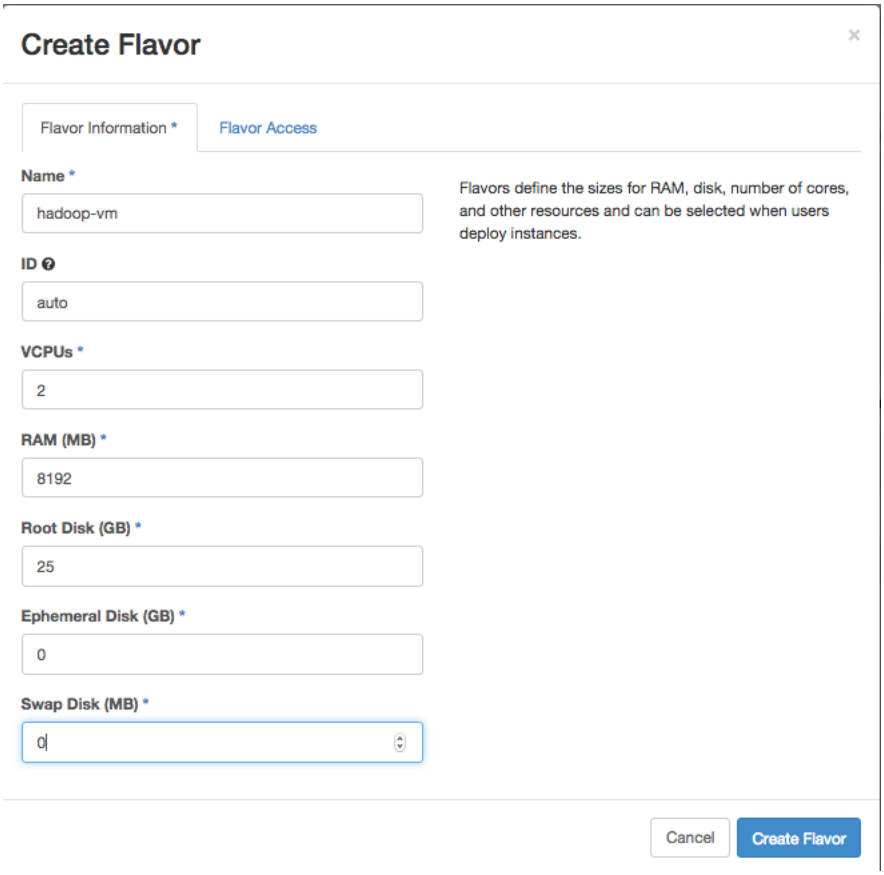
Şekil 4.6. Ubuntu 15.04 imajının sisteme eklenmesi



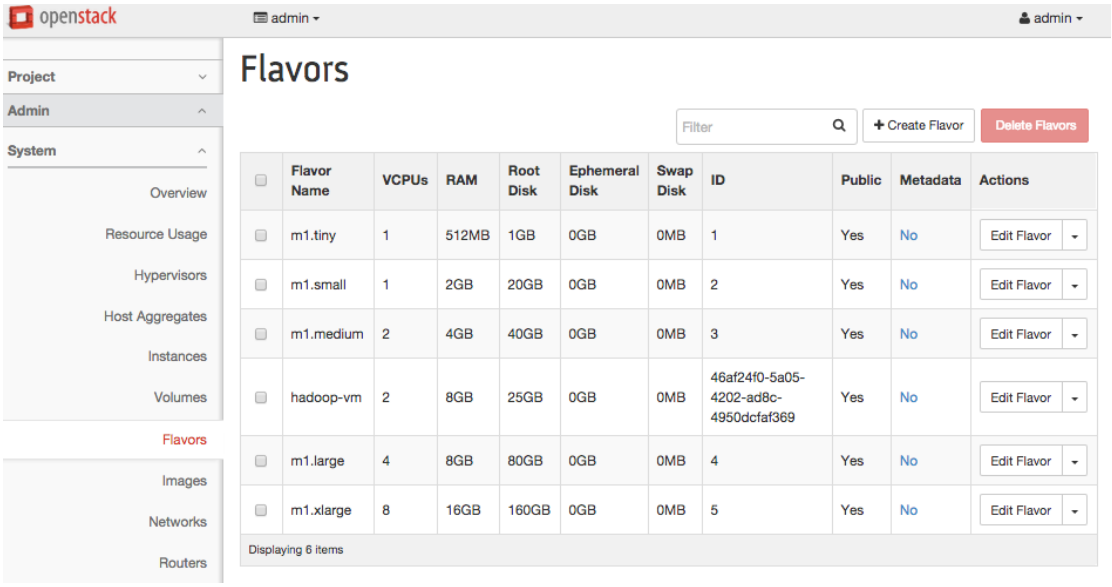
Şekil 4.7. Sisteme kayıtlı işletim sistemleri imajları

Sisteme eklenen işletim sistemi imajları kullanılarak kaynaklar izin verdiği sayıda kadar sanal makine oluşturulabilir. Ancak bu sanal makinelerin hangi özelliklere (CPU sayısı, RAM ve Hard Disk miktarı vb) sahip olacağına karar verilmesi gerekir. OpenStack, kurulacak sanal makinelere atanacak özellikleri Flavors (türler) adı verilen bir listede tutmaktadır.

Web arayüzünün Admin bölümünde, Flavors sayfasındaki bu listeye ekleme yapılabilir veya mevcut türler silinip değiştirilebilir. Bu tez kapsamında kurulan sanal makineler için oluşturulan hadoop-vm adındaki türün eklenmesi ekranı Şekil 4.5'te gösterilmiştir. Bu tür için 2 CPU, 8 GB RAM ve 25 GB hard disk seçilmiştir.



Şekil 4.8. Hadoop-vm sanal makine türünün tanımlanması



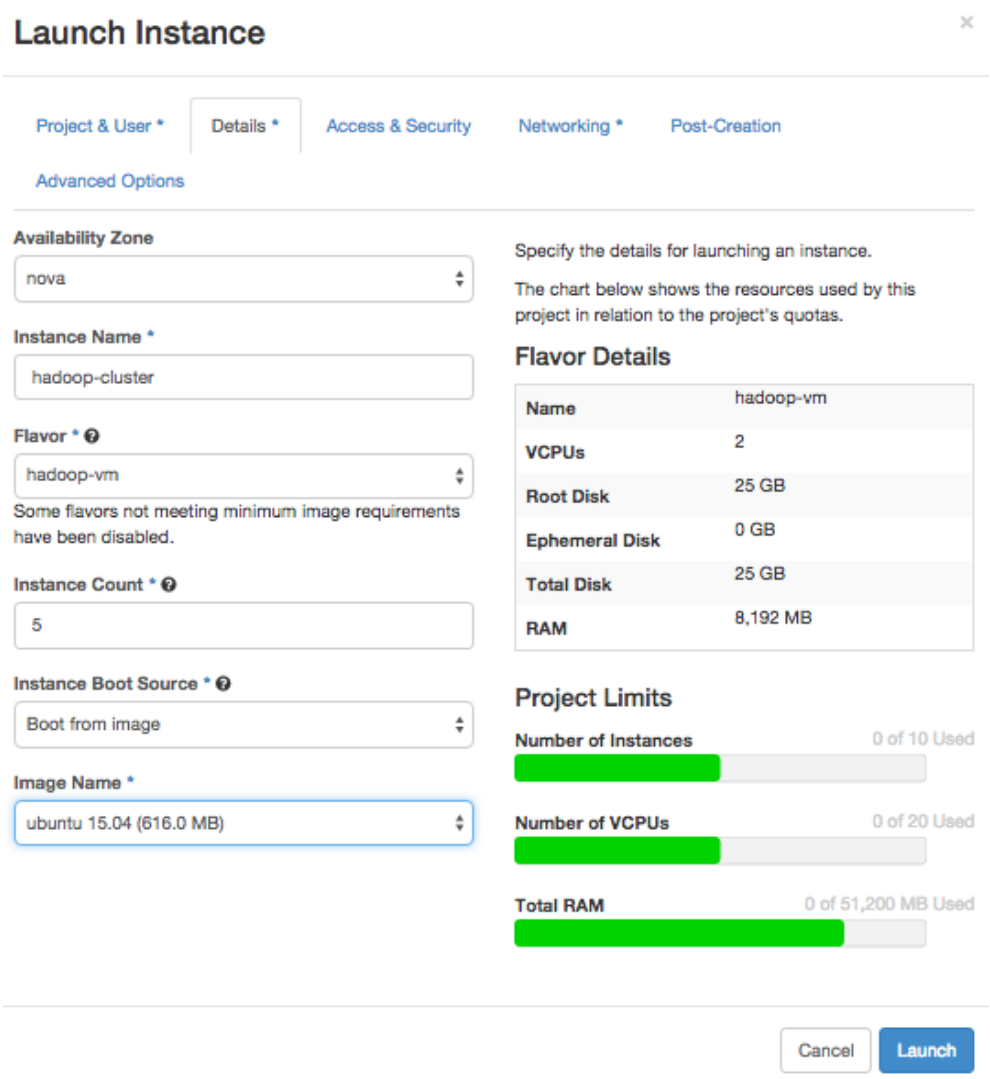
Şekil 4.9. Sistemde tanımlı sanal makine türleri listesi

OpenStack kurulumuna Key Pair, işletim sistemi imajı ve sanal makine türleri eklendikten sonra sanal makineler oluşturulabilir. Bunun için Projenin Instances sayfasındaki Launch Instance butonu kullanılır. Şekil 4.7'de gösterildiği gibi, açılan ekranda başlatılacak makine veya makinelerin özellikleri, flavor tipi, kullanılacak işletim sistemi imajı belirtilir.

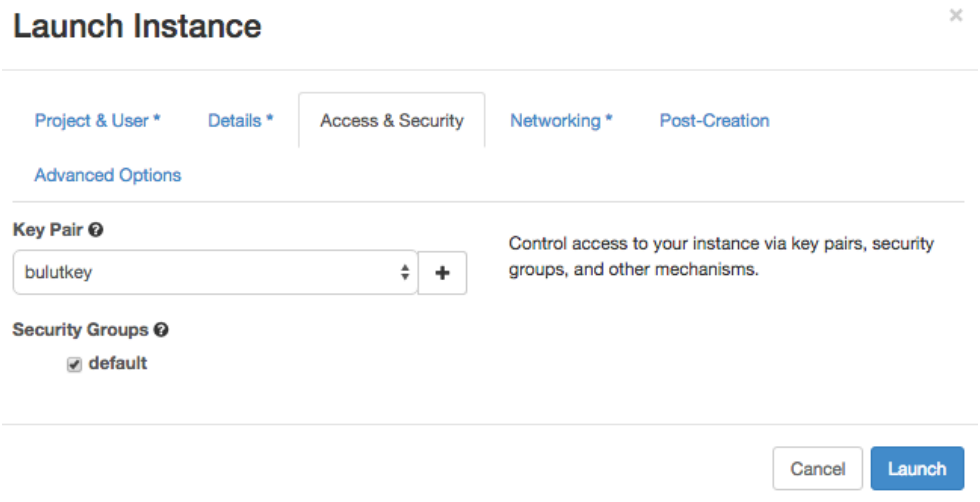
Aynı zamanda Şekil 4.8'de gösterildiği gibi kullanılacak Key Pair ve Şekil 4.9'da gösterildiği gibi makinelerin hangi ağlarda çalışacağının belirtilmesi gerekmektedir.

Gerekli parametrelerin girilmesinden sonra Launch butonuna basılınca istenildiği kadar sanal makine başlatılacak ve Instances ekranında Şekil 4.10'da olduğu gibi gösterilecektir.

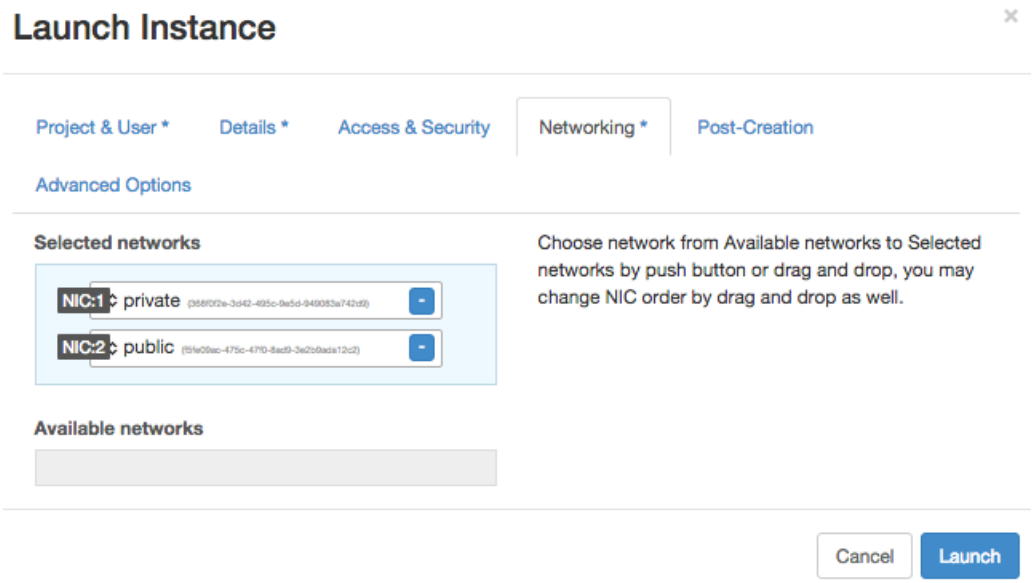
Başlatılan sanal makinelerle ilgili çeşitli işlemler yapılabilir. Makinelere Floating IP atanması, makinelerin yeniden başlatılması, durdurulması, kilitlenmesi, komut satırının görüntülenmesi, logların görüntülenmesi gibi işlemler listesi Şekil 4.11'de gösterilmiştir.



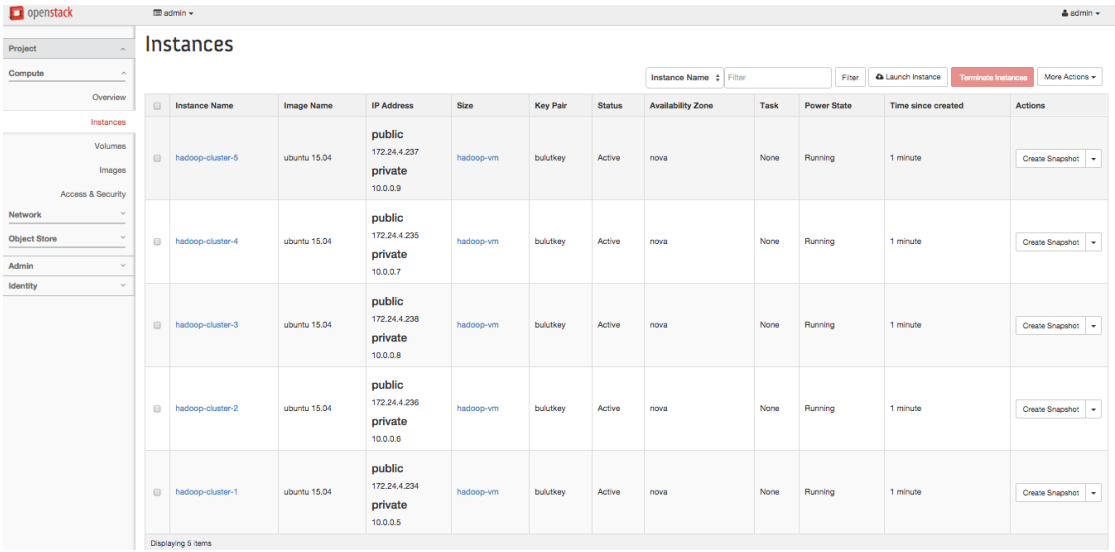
Şekil 4.10. Sanal makine başlatma ekranı – makine özelliklerinin belirlenmesi



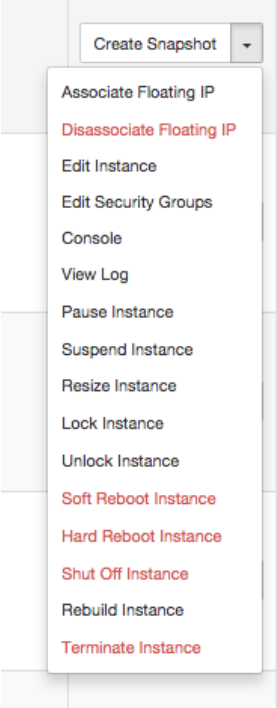
Şekil 4.11. Sanal makine başlatma ekranı – Key Pair seçilmesi



Şekil 4.12. Sanal makine başlatma ekranı – Ağ özelliklerinin seçilmesi



Şekil 4.13. Başlatılan sanal makine listesi



Şekil 4.14. Sanal makine işlemleri listesi

Görüldüğü gibi OpenStack, kolay kullanımlı web arayüzü aracılığıyla sanal makine ve bilgisayar kümeleri (cluster) oluşturulması gibi işlemleri son derece kolaylaştırmaktadır.

4.2. Apache Hadoop Kurulumu

Bu tez çalışması kapsamında Büyük Mekansal Veri analizi gerçekleştirebilmek için bulut.firat.edu.tr sunucusu üzerine kurulan OpenStack Bulut yazılımı ile sanal makineler oluşturulmuş ve bu makinelere 2 çekirdekli CPU, 8 GB RAM ve 25 GB hard disk verilmiştir. Makineler üzerinde CentOS 7 Linux işletim sistemi kurulumu gerçekleştirilmiştir.

Bu bölümde Hadoop 2.7.1 versiyonunun CentOS 7 üzerinde kurulum adımları verilmiştir. Aşağıda verilen her bir maddede, sırasıyla işletilen komutlar ve bu komutlara alınan cevaplar verilmiştir.

1- Öncelikle Java kurulumu yapılır. Sun JDK veya Open JDK alternatiflerinden birisi kurulabilir. Uygulamamızda OpenJDK tercih edilmiştir:

```
[bulut@bulut hadoop]$ sudo yum install java-1.7.0-openjdk
[sudo] password for bulut:
Loaded plugins: fastestmirror
```

```
...
Installed:
  java-1.7.0-openjdk.x86_64 1:1.7.0.85-2.6.1.2.el7_1

Dependency Installed:
  GConf2.x86_64 0:3.2.6-8.el7             giflib.x86_64 0:4.1.6-9.el7
java-1.7.0-openjdk-headless.x86_64 1:1.7.0.85-2.6.1.2.el7_1 javapackages-
tools.noarch 0:3.4.1-6.el7_0
  pcsc-lite-libs.x86_64 0:1.8.8-5.el7     python-javapackages.noarch 0:3.4.1-
6.el7_0 ttmkfdir.x86_64 0:3.0.9-41.el7                              tzdata-
java.noarch 0:2015f-1.el7
  xorg-x11-fonts-Type1.noarch 0:7.5-9.el7

Complete!
```

2- İşletim sistemine hadoop kullanıcısı eklenir, bu kullanıcının şifresi oluşturulur ve hadoop kullanıcısına geçilir:

```
[bulut@bulut hadoop]$ sudo useradd hadoop
[bulut@bulut hadoop]$ sudo passwd hadoop
Changing password for user hadoop.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.

[bulut@bulut hadoop]$ su - hadoop
Password:
```

3- hadoop kullanıcısı için SSH key oluşturulur:

```
[hadoop@bulut ~]$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
Generating public/private dsa key pair.
Created directory '/home/hadoop/.ssh'.
Your identification has been saved in /home/hadoop/.ssh/id_dsa.
Your public key has been saved in /home/hadoop/.ssh/id_dsa.pub.
The key fingerprint is:
10:c3:a4:59:ca:6a:02:7c:b5:9e:9f:1f:8e:b9:79:bf hadoop@bulut.firat.edu.tr
The key's randomart image is:
+--[ DSA 1024]----+
|     +=          |
|.  ..=oo         |
|.. .=..          |
|. ... ..         |
|. o  o  S        |
```

```
| o    . .        |
|       o .       |
|        *..      |
|       =oo.E.    |
+-----------------+
```

4- Oluşturulan SSH key yetkilendirilir ve key dosyasının izni 0600 olarak set edilir:

```
[hadoop@bulut ~]$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
[hadoop@bulut ~]$ chmod 0600 ~/.ssh/authorized_keys
```

5- Oluşturulan key ile localhost'a SSH yapılabildiği test edilir:

```
[hadoop@bulut ~]$ ssh localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is 1b:ca:a0:2e:30:a7:99:55:84:2c:4c:2f:8b:32:00:ef.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
Last login: Wed Aug 19 22:25:44 2015
[hadoop@bulut ~]$ exit
logout
Connection to localhost closed.
```

6- Ön hazırlık işlemlerinden sonra Hadoop kurulumuna geçilebilir. Öncelikle kurulmak istenen Hadoop versiyonunun indirilmesi gerekmektedir. Hadoop versiyonlarına http://apache.claz.org/hadoop/common adresinden ulaşılabilir. Bu tez çalışması için, mevcut en son Hadoop versiyonu olan 2.7.1 kurulmuştur:

```
    [hadoop@bulut ~]$ wget http://apache.claz.org/hadoop/common/hadoop-
2.7.1/hadoop-2.7.1.tar.gz

    --2015-08-19 22:31:32--  http://apache.claz.org/hadoop/common/hadoop-
2.7.1/hadoop-2.7.1.tar.gz
    Resolving apache.claz.org (apache.claz.org)... 74.63.227.45
    Connecting to apache.claz.org (apache.claz.org)|74.63.227.45|:80...
connected.
    HTTP request sent, awaiting response... 200 OK
    Length: 210606807 (201M) [application/x-gzip]
    Saving to: 'hadoop-2.7.1.tar.gz'
```

7- İndirilen sıkıştırılmış dosya açılır ve Hadoop klasörü oluşturulur:

```
[hadoop@bulut ~]$ ls
hadoop-2.7.1.tar.gz
```

```
[hadoop@bulut ~]$  tar xzf hadoop-2.7.1.tar.gz
[hadoop@bulut ~]$ ls
hadoop-2.7.1  hadoop-2.7.1.tar.gz
```

8- Hadoop çalışması için bir takım ortam değişkenlerinin set edilmesi gerekmektedir. Bunun için hadoop kullanıcısının ortam değişkenlerinin tanımlandığı .bashrc dosyası aşağıdaki şekilde güncelenir:

```
[hadoop@bulut ~]$ nano .bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=
# User specific aliases and functions
export JAVA_HOME=/usr/lib/jvm/jre
export HADOOP_HOME=/home/hadoop/hadoop-2.7.1
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export JAVA_LIBRARY_PATH=$HADOOP_HOME/lib/native:$JAVA_LIBRARY_PATH
```

9- .bashrc dosyasında tanımlanan ortam değişkenlerinin etkin olabilmesi için bu dosyanın source yapılması gerekir.

```
[hadoop@bulut ~]$ source $HOME/.bashrc
```

10- Hadoop programının kurulum yapılan makinede çalışabilmesi için konfigürasyon dosyalarında bazı eklemeler yapılması gerekmektedir. Öncelikle hadoop klasörüne gidilir ve dosyalar verilen şekilde güncellenir.

```
[hadoop@bulut ~]$ cd hadoop-2.7.1
[hadoop@bulut hadoop-2.7.1]$ nano etc/hadoop/core-site.xml

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
```

```
<configuration>
<property>
  <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
</property>
</configuration>

[hadoop@bulut hadoop-2.7.1]$ nano etc/hadoop/hdfs-site.xml

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
 <name>dfs.replication</name>
 <value>1</value>
</property>

<property>
  <name>dfs.name.dir</name>
    <value>file:///home/hadoop/hadoopdata/hdfs/namenode</value>
</property>

<property>
  <name>dfs.data.dir</name>
    <value>file:///home/hadoop/hadoopdata/hdfs/datanode</value>
</property>
</configuration>

[hadoop@bulut      hadoop-2.7.1]$      cp      $HADOOP_HOME/etc/hadoop/mapred-
site.xml.template $HADOOP_HOME/etc/hadoop/mapred-site.xml

[hadoop@bulut hadoop-2.7.1]$ nano etc/hadoop/mapred-site.xml

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>mapreduce.framework.name</name>
   <value>yarn</value>
</property>
</configuration>

[hadoop@bulut hadoop-2.7.1]$ nano etc/hadoop/yarn-site.xml

<?xml version="1.0"?>
<configuration>
<property>
  <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
</configuration>
```

11- Hadoop ortam değişkenleri dosyasında JAVA_HOME değişkeninin kurulumu yapılan Java klasörünü gösterecek şekilde güncellenmesi gerekmektedir:

```
[hadoop@bulut hadoop-2.7.1]$ nano etc/hadoop/hadoop-env.sh
```

```
export JAVA_HOME=/usr/lib/jvm/jre
```

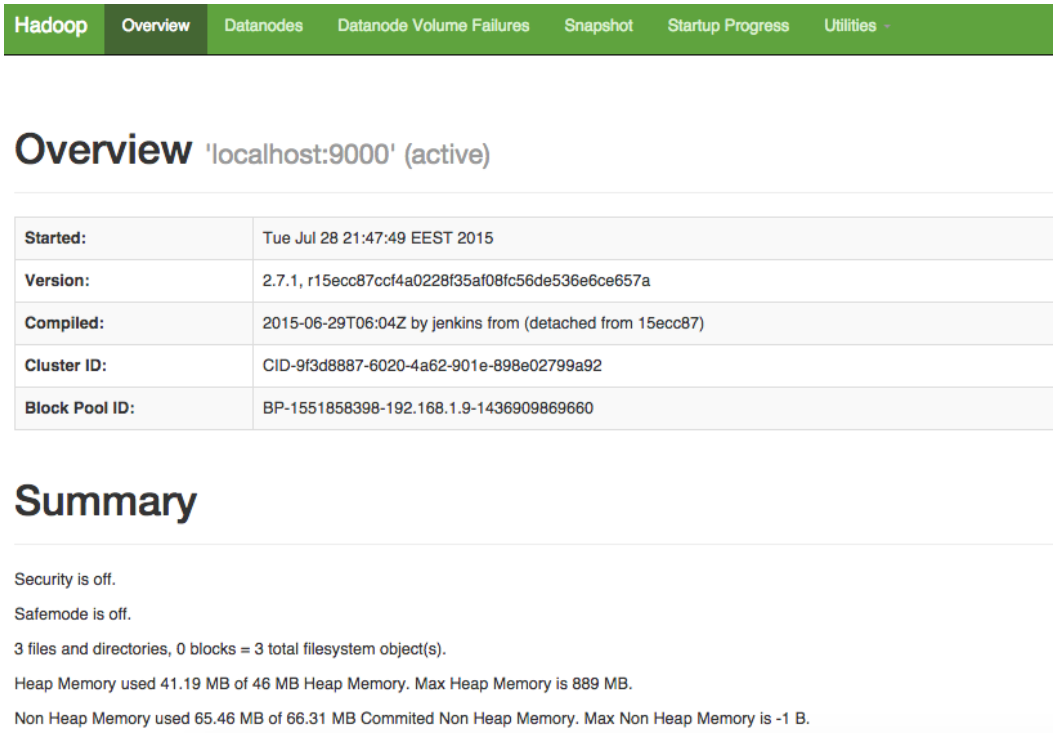
12- Hadoop konfigürasyonunun bu şekilde tamamlanmış olur. Şimdi hadoop çalıştırılması ve test edilmesi gerçekleştirilebilir. Öncelikle dağıtık dosya sistemi formatlanır ve kullanıma hazır hale getirilir:

```
[hadoop@bulut hadoop-2.7.1]$ hdfs namenode format
15/08/19 22:45:52 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = bulut.firat.edu.tr/10.1.1.27
STARTUP_MSG:   args = [format]
STARTUP_MSG:   version = 2.7.1

....

15/08/19 22:45:52 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at bulut.firat.edu.tr/10.1.1.27
************************************************************/
```

13- hadoop/sbin klasöründeki start-dfs.sh scripti yardımıyla dağıtık dosya sistemi başlatılır:

```
[hadoop@bulut hadoop-2.7.1]$ ./sbin/start-dfs.sh
    Starting namenodes on [localhost]
    localhost: starting namenode, logging to /home/hadoop/hadoop-
2.7.1/logs/hadoop-hadoop-namenode-bulut.firat.edu.tr.out
    localhost: starting datanode, logging to /home/hadoop/hadoop-
2.7.1/logs/hadoop-hadoop-datanode-bulut.firat.edu.tr.out
    Starting secondary namenodes [0.0.0.0]
    The authenticity of host '0.0.0.0 (0.0.0.0)' can't be established.
    ECDSA key fingerprint is 1b:ca:a0:2e:30:a7:99:55:84:2c:4c:2f:8b:32:00:ef.
    Are you sure you want to continue connecting (yes/no)? yes
    0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known
hosts.
    0.0.0.0: starting secondarynamenode, logging to /home/hadoop/hadoop-
2.7.1/logs/hadoop-hadoop-secondarynamenode-bulut.firat.edu.tr.out
```

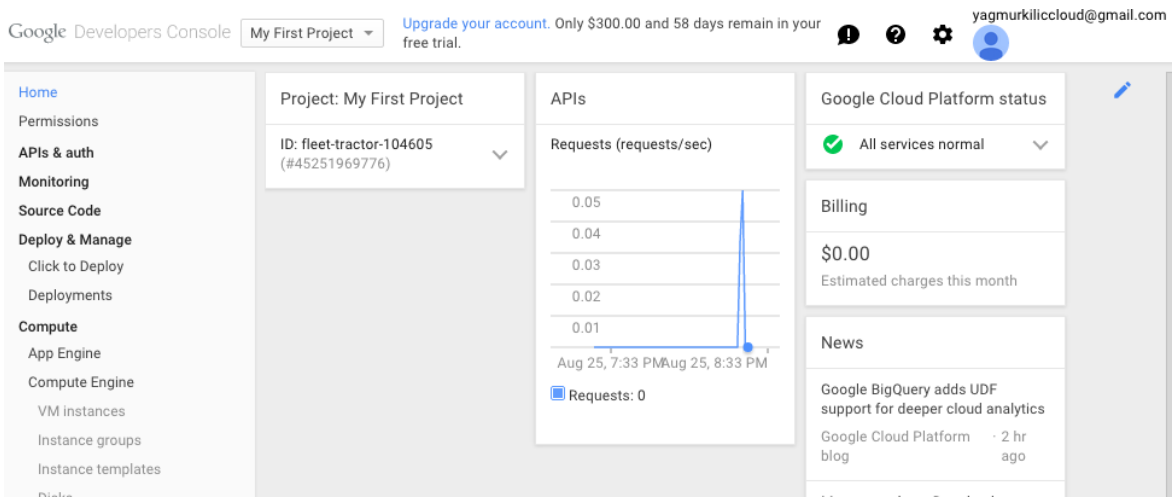
Böylelikle Hadoop kurulumu tamamlanmış olur. Kurulum adımları işletim sistemine ve Hadoop versiyonuna göre farklılık gösterebilmektedir. Kurulumu yapılan Hadoop sisteminin çeşitli özelliklerine ve dağıtık dosya sistemine web arayüzü ile ulaşılabilmektedir. Web arayüzüne http://localhost:50070/ adresinden ulaşılabilir.



Şekil 4.15. Hadoop Web Arayüzü

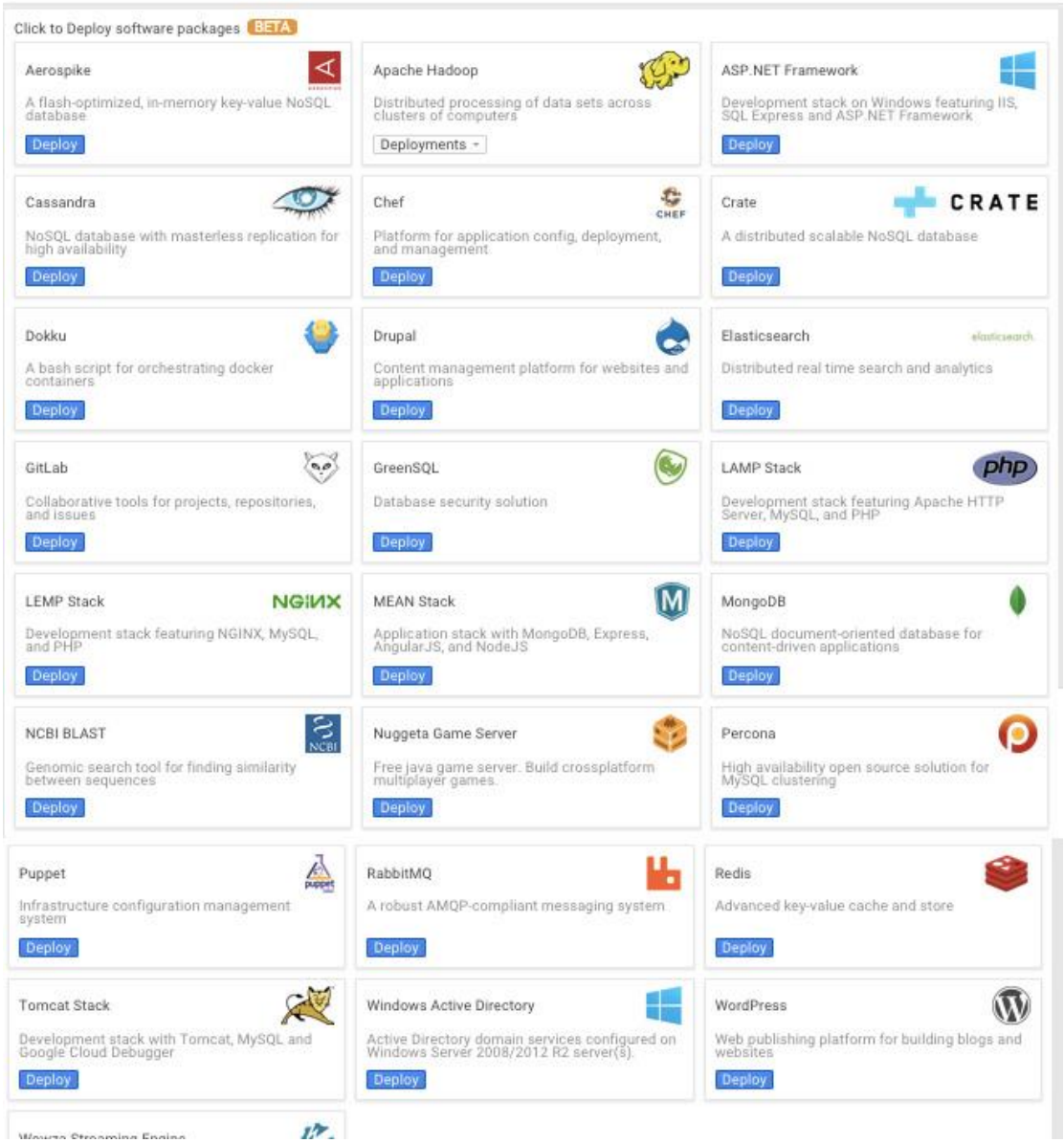
4.3. Google Cloud Üzerinde Apache Hadoop Kümesi Kurulumu

Google Cloud hesabı oluşturulup giriş yapıldıktan sonra, hesabın genel görünümünü gösteren bir ekran gelir. Bu ekranda kullanılan kaynaklar, güncel fatura, kullanım istatistikleri gibi bilgiler gösterilmektedir.



Şekil 4.16. Google Cloud kullanıcı hesabı özet sayfası

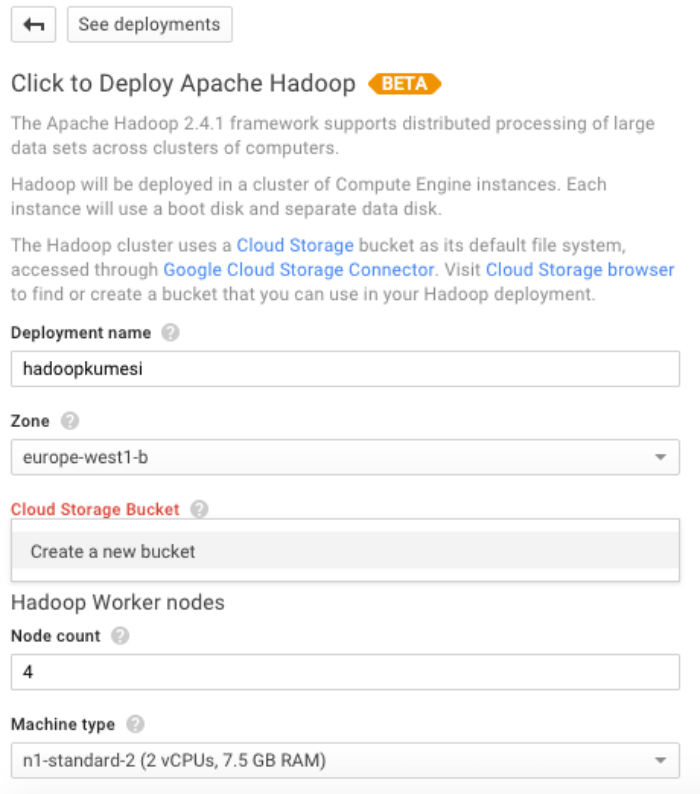
Sayfanın sol tarafında yapılabilecekleri gösteren menüler bulunmaktadır. Kullanıcı, bu menüler yardımıyla sanal makine, depolama alanları vb oluşturma ve yönetme araçlarına ulaşır. Ayrıca "Deploy & Manage" menüsü altında da tek tuşla birçok farklı uygulamanın kurulması için kısayollar bulunmaktadır.



Şekil 4.17. Google Cloud "Click to Deploy" sayfasında kurulabilecek yazılımlar.

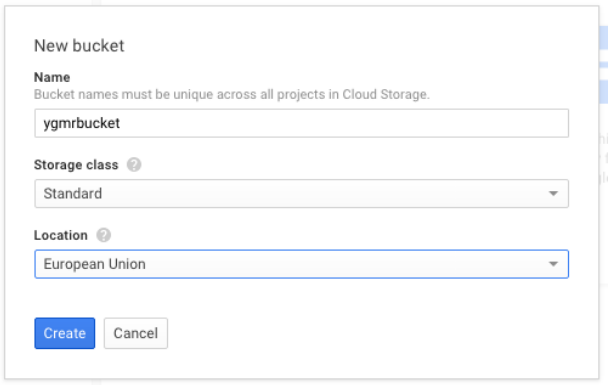
Tek tuşla kurulum yapılabilecek yazılımlardan birisi de Apache Hadoop'tur. Aşağıdaki resimlerle Google Cloud üzerinde Apache Hadoop kurulum adımları verilmiştir.

1- Apache Hadoop Deploy butonuna tıklanınca aşağıdaki şekilde gösterilen Hadoop kümesinin özelliklerinin belirlendiği ekran gelmektedir.



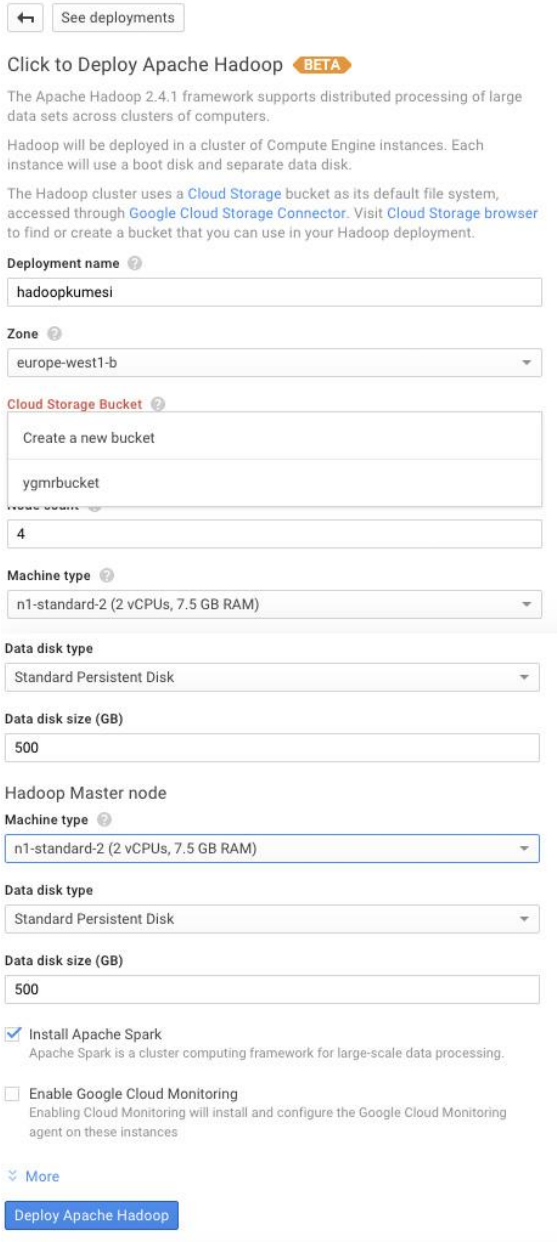
Şekil 4.18. Hadoop Kurulum ekranı

2- Hadoop kümesi varsayılan dosya sistemi olarak "Google Cloud Storage" kullandığı için ilk adım olarak bir "Storage Bucket" oluşturulması gerekmektedir. Dolayısıyla "Create a new bucket" seçilir. Açılan sayfada aşağıda gösterilen bilgiler doldurularak yeni bir bucket oluşturulur.



Şekil 4.19. Storage Bucket oluşturma ekranı

3- Storage bucket oluşturulduktan sonra ana kurulum ekranında bu bucket seçilir ve kümenin diğer özellikleri seçilir.



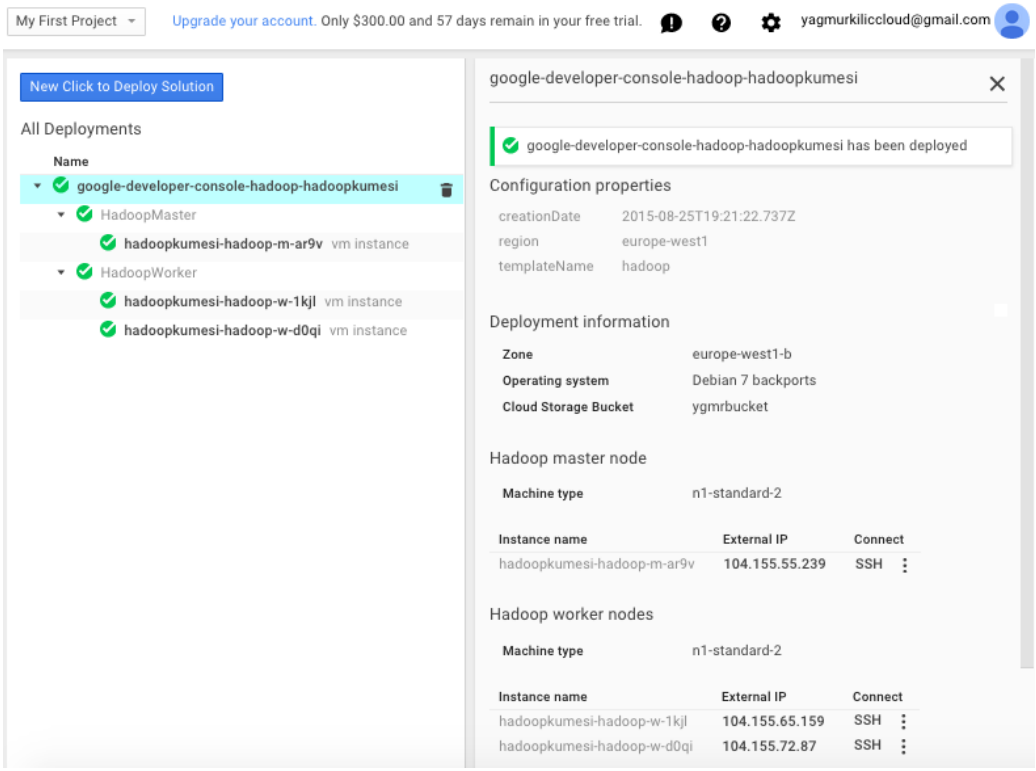
Şekil 4.20. Hadoop Kümesi kurulum ekranı

Burada öncelikle kümeye bir isim verilir. Hadoop kümesinin ABD, Avrupa veya Doğu Asyadaki Google sunucu merkezlerinden hangisinde kurulacağı ve oluşturulan bucket seçilir.

Daha sonra hadoop kümesinde kaç tane işçi düğümün (Worker Node) olacağı ve bu düğümlerin türü (CPU sayısı ve RAM miktarı) belirlenir.

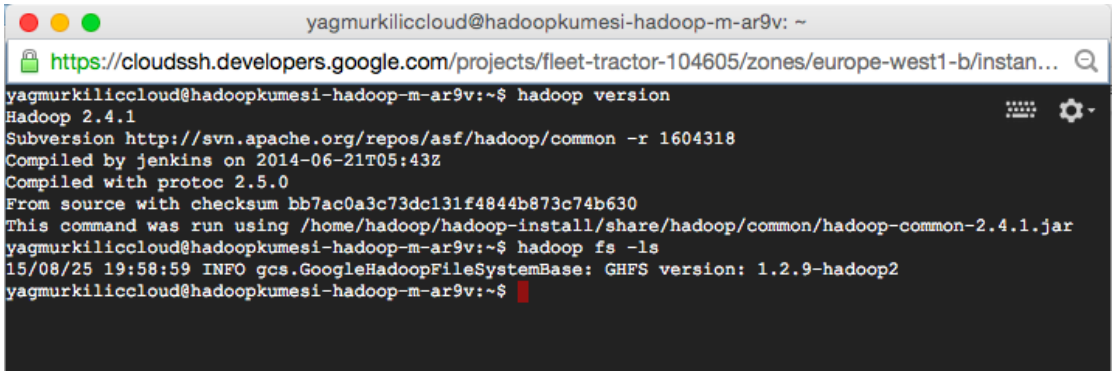
Ayrıca Hadoop Master düğümün özellikleri de benzer şekilde seçildikten sonra "Deploy Apache Hadoop" butonu yardımıyla kurulum tamamlanır. Kurulum

tamamlandıktan sonra Deployments linki yardımıyla, kurulumu tamamlanan Hadoop kümesinin özellikleri (IP adresleri, SSH erişim linkleri vb) görüntülenebilir.



Şekil 4.21. Kurulumu tamamlanan Hadoop kümesinin özellikleri

4- Hadoop master node bölümündeki SSH butonuna tıklanması ile master düğüme SSH erişimi yapılabilecek bir pencere açılır. Bu pencere yardımıyla çalışır vaziyetteki Hadoop kümesi kontrol edilebilir.



Şekil 4.22. Hadoop master node SSH komut satırı ekranı

4.4. Google Cloud - Hadoop Kurulumu üzerinde örnek MapReduce programı çalıştırılması

Bu kısımda, önceki bölümde anlatılan, Hadoop kümesi üzerinde basit bir MapReduce işinin nasıl gerçekleştirildiği anlatılmıştır. Bunun için internetteki örnek bir web sayfası indirilmiş ve bu sayfada geçen kelimelerin sayılması işlemi gerçekleştirilmiştir. Bu örnek için takip edilen adımlar şunlardır:

1- web sayfası indirilir

```
yagmurkiliccloud@hadoopkume-hadoop-m-1-vm:/home/hadoop/hadoop-install$
hadoop version
Hadoop 2.4.1

yagmurkiliccloud@hadoopkume-hadoop-m-1-vm:~$ curl
http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-
common/ClusterSetup.html > /tmp/setup.html
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 54148  100 54148    0     0  91578      0 --:--:-- --:--:-- --:--:--  120k
```

2- HDFS üzerinde input klasörü oluşturulur:

```
yagmurkiliccloud@hadoopkume-hadoop-m-1-vm:~$ hadoop fs -mkdir input/

15/08/30 18:59:27 INFO gcs.GoogleHadoopFileSystemBase: GHFS version: 1.2.9-
hadoop2
```

3- indirilen web sayfası HDFS üzerindeki input klasörüne kopyalanır:

```
yagmurkiliccloud@hadoopkume-hadoop-m-1-vm:~$ hadoop fs -copyFromLocal
/tmp/setup.html input/

15/08/30 19:03:10 INFO gcs.GoogleHadoopFileSystemBase: GHFS version: 1.2.9-
hadoop2
```

indirilen dosyanın HDFS'e kopyalandığı kontrol edilebilir:

```
yagmurkiliccloud@hadoopkume-hadoop-m-1-vm:~$ hadoop fs -ls input/

15/08/30 19:03:42 INFO gcs.GoogleHadoopFileSystemBase: GHFS version: 1.2.9-
hadoop2
```

```
Found 1 items
-rwx------   3 yagmurkiliccloud yagmurkiliccloud      54148 2015-08-30 19:03
../input/setup.html
```

4- hadoop kurulum klasörüne gidilir:

```
yagmurkiliccloud@hadoopkume-hadoop-m-1-vm:~$ cd /home/hadoop/hadoop-install/
yagmurkiliccloud@hadoopkume-hadoop-m-1-vm:/home/hadoop/hadoop-install$ ls
bin  Hadoop_2.4.1-Linux-amd64-64.tar.gz  lib      LICENSE.txt  NOTICE.txt  sbin
etc  include                            libexec  logs         README.txt  share
```

5- Hadoop kurulumu ile gelen örnek kelime sayma programını içeren jar dosyası input klasörü üzerinde çalıştırılır

```
yagmurkiliccloud@hadoopkume-hadoop-m-1-vm:/home/hadoop/hadoop-install$ hadoop jar
share/hadoop/mapreduce/hadoop-*-e
xamples-*.jar wordcount input output

15/08/30 19:06:06 INFO gcs.GoogleHadoopFileSystemBase: GHFS version: 1.2.9-
hadoop2
15/08/30 19:06:07 INFO client.RMProxy: Connecting to ResourceManager at
hadoopkume-hadoop-m-1-vm/10.240.95.17:8032
15/08/30 19:06:20 INFO input.FileInputFormat: Total input paths to process : 1
15/08/30 19:06:28 INFO mapreduce.JobSubmitter: number of splits:1
15/08/30 19:06:30 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1440960695542_0001
15/08/30 19:06:32 INFO impl.YarnClientImpl: Submitted application
application_1440960695542_0001
15/08/30 19:06:32 INFO mapreduce.Job: The url to track the job:
http://hadoopkume-hadoop-m-1-vm:8088/proxy/applicat
ion_1440960695542_0001/
15/08/30 19:06:32 INFO mapreduce.Job: Running job: job_1440960695542_0001
15/08/30 19:06:58 INFO mapreduce.Job: Job job_1440960695542_0001 running in uber
mode : false
15/08/30 19:06:58 INFO mapreduce.Job:  map 0% reduce 0%
15/08/30 19:07:05 INFO mapreduce.Job:  map 100% reduce 0%
15/08/30 19:07:19 INFO mapreduce.Job:  map 100% reduce 13%
15/08/30 19:07:22 INFO mapreduce.Job:  map 100% reduce 25%
15/08/30 19:07:23 INFO mapreduce.Job:  map 100% reduce 38%
15/08/30 19:07:36 INFO mapreduce.Job:  map 100% reduce 71%
15/08/30 19:07:37 INFO mapreduce.Job:  map 100% reduce 96%
15/08/30 19:07:40 INFO mapreduce.Job:  map 100% reduce 100%
15/08/30 19:08:06 INFO mapreduce.Job: Job job_1440960695542_0001 completed
successfully
15/08/30 19:08:06 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=32739
                FILE: Number of bytes written=910841
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                GS: Number of bytes read=54245
                GS: Number of bytes written=27430
                GS: Number of read operations=0
                GS: Number of large read operations=0
                GS: Number of write operations=0
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=8
                Rack-local map tasks=1
```

```
                Total time spent by all maps in occupied slots (ms)=5782
                Total time spent by all reduces in occupied slots (ms)=182822
                Total time spent by all map tasks (ms)=5782
                Total time spent by all reduce tasks (ms)=182822
                Total vcore-seconds taken by all map tasks=5782
                Total vcore-seconds taken by all reduce tasks=182822
                Total megabyte-seconds taken by all map tasks=5920768
                Total megabyte-seconds taken by all reduce tasks=187209728
        Map-Reduce Framework
                Map input records=1424
                Map output records=4318
                Map output bytes=62949
                Map output materialized bytes=32739
                Input split bytes=97
                Combine input records=4318
                Combine output records=1332
                Reduce input groups=1332
                Reduce shuffle bytes=32739
                Reduce input records=1332
                Reduce output records=1332
                Spilled Records=2664
                Shuffled Maps =8
                Failed Shuffles=0
                Merged Map outputs=8
                GC time elapsed (ms)=1234
                CPU time spent (ms)=8380
                Physical memory (bytes) snapshot=2026803200
                Virtual memory (bytes) snapshot=6650531840
                Total committed heap usage (bytes)=1338507264
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=54148
        File Output Format Counters
                Bytes Written=27430
```

6- Kelime sayma programı çalıştıktan sonra oluşturulan sonuç dosyaları listelenir

```
yagmurkiliccloud@hadoopkume-hadoop-m-1-vm:/home/hadoop/hadoop-install$ hadoop fs
-ls output/

15/08/30 19:19:15 INFO gcs.GoogleHadoopFileSystemBase: GHFS version: 1.2.9-
hadoop2
Found 9 items
-rwx------   3 yagmurkiliccloud yagmurkiliccloud          0 2015-08-30 19:07
../output/_SUCCESS
-rwx------   3 yagmurkiliccloud yagmurkiliccloud       3648 2015-08-30 19:07
../output/part-r-00000
-rwx------   3 yagmurkiliccloud yagmurkiliccloud       3552 2015-08-30 19:07
../output/part-r-00001
-rwx------   3 yagmurkiliccloud yagmurkiliccloud       3495 2015-08-30 19:07
../output/part-r-00002
-rwx------   3 yagmurkiliccloud yagmurkiliccloud       3015 2015-08-30 19:07
../output/part-r-00003
-rwx------   3 yagmurkiliccloud yagmurkiliccloud       3490 2015-08-30 19:07
../output/part-r-00004
-rwx------   3 yagmurkiliccloud yagmurkiliccloud       3722 2015-08-30 19:07
../output/part-r-00005
```

```
-rwx------   3 yagmurkiliccloud yagmurkiliccloud       3095 2015-08-30 19:07
../output/part-r-00006
-rwx------   3 yagmurkiliccloud yagmurkiliccloud       3413 2015-08-30 19:07
../output/part-r-00007
```

7- Sonuç dosyalarından birisinin içeriğine bakma:

```
yagmurkiliccloud@hadoopkume-hadoop-m-1-vm:/home/hadoop/hadoop-install$ hadoop fs
-cat output/part-r-00000

15/08/30 19:19:46 INFO gcs.GoogleHadoopFileSystemBase: GHFS version: 1.2.9-
hadoop2
$HADOOP_PREFIX/sbin/stop-dfs.sh 1
/>      5
1000    1
10020.  1
</body> 1
<h5>Auth</h5>   1
<img    3
<li     94
<meta   2
<p>For  1
<p>Installing   1
<p>Many 1
<p>The  4
<pre>[yarn]$    8
```

8- Sonuç dosyalarının tamamı birleştirilip tek bir dosya olarak lokal dosya sistemine kopyalanabilir:

```
yagmurkiliccloud@hadoopkume-hadoop-m-1-vm:/home/hadoop/hadoop-install$ hadoop fs
-getmerge output/ /tmp/output.txt
```

9- İşlemler bittikten sonra input ve output klasörlerinin silinebilir:

```
hadoop fs -rm -r output/
hadoop fs -rm -r input/
```

4.5. Google Cloud Üzerindeki Hadoop Kümesinde Apache Mahout ile Kümeleme Örneği

Apache Mahout kullanarak kümeleme işleminin nasıl yapıldığını anlamak amacıyla öncelikle basit bir kümeleme örneği çalıştırılmıştır. Bu örnekte Gauss dağılımına göre üretilmiş sayılar, K-Means algoritmasına göre toplam 4 adet kümeye düşecek şekilde kümelenmiştir. Bu bölümde Mahout ile örnek kümeleme için takip edilen adımlar verilmiştir.

Aşağıda rastgele Gauss sayıları üreten program verilmiştir.

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

public final class RandomGaussian {

    private Random fRandom = new Random();
    private FileWriter fw;

    public RandomGaussian() {
        try {
            fw = new FileWriter("gauss1.txt");
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    private double getGaussian(double aMean, double aVariance) {
        return aMean + fRandom.nextGaussian() * aVariance;
    }

    public void gaussDosyaOlustur(int satirSayisi) {
        try {
            double MEAN = -0f;
            double VARIANCE = 1f;
            for (int i = 1; i <= satirSayisi; ++i) {
                this.fw.write(i + "," + getGaussian(MEAN, VARIANCE) + ","
+ getGaussian(MEAN, VARIANCE) + "\n");
                fw.flush();
            }
            fw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String... aArgs) {
        RandomGaussian g = new RandomGaussian();
        g.gaussDosyaOlustur(250);
    }
}
```

Rasgele Gauss sayısı üreten ve dosyaya yazan Java uygulaması

1- Yukarıda verilen program kullanılarak 250 adet gauss sayısı çifti üretilmiş ve Şekil 4A'da verilen formatta, *gauss.txt* adında bir dosyaya kaydedilmiştir.

```
1,1.6962038293741832,0.2803405088611849
2,-0.570820998814006,-0.2874639500955714
3,0.24794238679199598,-2.0254547319223404
4,-1.1556005255236366,1.3002204339111034
5,0.08975995158226786,-1.0514300623917394
```

Şekil 4.23. Örnek Gauss dosyası formatı

2- Mahout, analiz edilecek verilerin SequenceFile adındaki bir dosya formatında olmasını gerektirir. Dolayısıyla üretilen metin dosyasının Mahout tarafından kümeleme için

kullanılabilmesi amacıyla aşağıda verilen SequenceDosyaYap sınıfı kullanılmıştır. Program input olarak aldığı dosyayı okuyup sequence vektör formatına dönüştürür.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;
import org.apache.mahout.math.DenseVector;
import org.apache.mahout.math.VectorWritable;

class SequenceDosyaYap {

    private SequenceDosyaYap() {
    }

    public static final int SUTUN_SAYISI = 2;

    public void seqDosya() throws IOException {
        try {
            String GIRIS_DOSYA = "gauss.txt";
            String SONUC_DOSYA = "gauss";
            int DOSYA_SATIR_SAYISI = 100000;
            List<VectorWritable> liste = new ArrayList<>();

            BufferedReader br = null;
            br = new BufferedReader(new FileReader(GIRIS_DOSYA));
            String sCurrentLine;
            while ((sCurrentLine = br.readLine()) != null) {
                double[] sutunlar = new double[SUTUN_SAYISI - 1];
                for (int indx = 1; indx < SUTUN_SAYISI; ++indx) {
                    sutunlar[indx - 1] =
Double.parseDouble(sCurrentLine.split(",")[indx]);
                }
                liste.add(new VectorWritable(new DenseVector(sutunlar)));
            }
            Configuration conf = new Configuration();
            FileSystem fs = FileSystem.get(conf);
            Path path = new Path(SONUC_DOSYA);
            String yol = path.toString();

            int file_no = 0;
            path = new Path(yol + "_" + file_no++);
            SequenceFile.Writer writer = new SequenceFile.Writer(fs, conf,
path, Text.class, VectorWritable.class);
            VectorWritable vec = new VectorWritable();
            int i = 0;

            for (VectorWritable vw : liste) {
                //dosyaya satırı yaz
                writer.append(new Text("ornek_" + i++), vw);
                //max satır sayısına ulaşıldığında yeni bir dosya oluştur
                if (i % DOSYA_SATIR_SAYISI == 0) {
                    writer.close();
                    i = 0;
                    path = new Path(yol + "_" + file_no++);
                    writer = new SequenceFile.Writer(fs, conf, path,
Text.class, VectorWritable.class);
                    vec = new VectorWritable();
```

```
                }
            }
            writer.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) throws Exception {
        SequenceDosyaYap v = new SequenceDosyaYap();
        v.seqDosya();
    }
}
```

Mahout Sequence dosyası oluşturma programı

3- Oluşturulan Sequence vektör dosyasının Mahout tarafından analiz edilebilmesi için HDFS'te bulunması gerekir. Dolsyısıyla bu dosya HDFS'de oluşturulan input klasörünün altına atılır:

```
hadoop@hadoopkume-hadoop-m-1-vm:~/input$ hadoop fs -put gauss input/gauss

15/08/30 20:38:10 INFO gcs.GoogleHadoopFileSystemBase: GHFS version: 1.2.9-
hadoop2
```

4- HDFS'te depolanan gauss dosyası üzerinde K-Means kümeleme algoritması çalıştırılır:

```
hadoop@hadoopkume-hadoop-m-1-vm:~/apache-mahout-distribution-0.10.1$ bin/mahout
kmeans -i input/gauss -c /cluster
 -o /seqfiles-final-cluster -x 1000 -k 4 -ow --clustering -cd 0.5
```

5- K-Means kümeleme sonucu oluşan sonuç dosyaları *clusterdump* komutu yardımıyla yerel dosya sistemine kopyalanıp içeriğine bakılabilir:

```
hadoop@hadoopkume-hadoop-m-1-vm:~/apache-mahout-distribution-0.10.1$ bin/mahout
clusterdump -i /seqfiles-final-clus
ter/clusters-1-final -p /seqfiles-final-cluster/clusteredPoints -o clusters.txt

hadoop@hadoopkume-hadoop-m-1-vm:~/apache-mahout-distribution-0.10.1$ more
clusters.txt

{"r":[0.354,0.588],"c":[-1.238,0.642],"n":45,"identifier":"VL-195"}
        Weight : [props - optional]:  Point:
        1.0 : [distance=0.32489806150509537]: [-1.405,0.097]
        1.0 : [distance=0.6773672631360514]: [-0.6,0.123]
        1.0 : [distance=0.0039483488058489336]: [-1.187,0.605]
        1.0 : [distance=0.036039537814619216]: [-1.284,0.826]
        1.0 : [distance=0.8377703098617513]: [-0.504,1.189]
        1.0 : [distance=3.01080580329981]: [-2.489,-0.561]
        1.0 : [distance=0.1819403061928173]: [-0.904,0.907]
        1.0 : [distance=0.7360934873040197]: [-1.298,1.498]
```

```
1.0 : [distance=0.5273690579101049]: [-0.712,0.141]
```

6- Sonuç dosyasında kümeleme sonucunu anlamak kolay olmadığından, Mahout, grafiksel olarak da bu sonuçları verebilmektedir. Bunun için *graphml* formatında çıktı almak mümkündür. Bunun için aşağıdaki komut kullanılabilir:

bin/mahout clusterdump -i /seqfiles-final-cluster/clusters-1-final -p /seqfiles-final-cluster/clusteredPoints -of GRAPH_ML -o gauss.graphml

Bu komutun çıktısı olarak üretilen graphml dosyası XML dili ile yazılan ve kenar ve köşelerden oluşan bir graf tanımlama dosyasıdır. Aşağıda dosya içeriğini gösteren bir örnek verilmiştir.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
    <key attr.name="r" attr.type="int" for="node" id="r"/>
    <key attr.name="g" attr.type="int" for="node" id="g"/>
    <key attr.name="b" attr.type="int" for="node" id="b"/>
    <key attr.name="size" attr.type="int" for="node" id="size"/>
    <key attr.name="weight" attr.type="float" for="edge" id="weight"/>
    <key attr.name="x" attr.type="float" for="node" id="x"/>
    <key attr.name="y" attr.type="float" for="node" id="y"/>
    <graph edgedefault="undirected">
        <node id="220_">
            <data key="r">20</data>
            <data key="g">0</data>
            <data key="b">0</data>
            <data key="x">1000.0</data>
            <data key="y">0.0</data>
        </node>
        <node id="ornek_0">
            <data key="r">20</data>
            <data key="g">0</data>
            <data key="b">0</data>
            <data key="x">995.87555</data>
            <data key="y">23.39091</data>
        </node>
        <edge id="220__ornek_0" source="220_" target="ornek_0">
            <data key="weight">23.75175247008121</data>
        </edge>
    </graph>
</graphml>
```

Örnek Graph ML dosyası

7- Kümeleme sonuçlarını veren gauss.graphml dosyası Gephi adındaki açık kaynak grafik programı ile görselleştirilebilir.

4.6. Mekansal Veri Kümeleme

Bu tezde mekânsal verilerin kümelenmesi işlemi için Apache Mahout tarafından sunulan K-Means algoritması kullanılmıştır.

Kümeleme testleri için, www.naviskop.com adresinde çalışan Naviskop araç takip sistemi tarafından bir yıl süreyle toplanan ve takip edilen araçların konumunu barındıran log dosyalarından çıkarılan konum verileri kullanılmıştır.

Ancak kullanılan veriler GPS verileri olduğu için koordinatlar WGS-84 koordinat sistemindedir ve GPS cihazından alındığı şekliyle kümeleme işlemlerinde kullanılamaz. Çünkü bu koordinatlar elipsoid şeklinde olduğu kabul edilen dünya üzerindeki noktaları ifade etmektedir ve yeryüzünün herhangi farklı iki noktasında, enlem ve boylamdaki aynı miktardaki değişim aynı uzunluklara denk gelmez. Dolayısıyla öncelikle coğrafi koordinatların düzlem üzerinde kabul edilebilecek bir Kartezyen koordinat sistemine dönüştürülmesi gerekir.

Bu dönüşüm aşağıdaki şekilde gerçekleştirilebilir:

- Elimizde coğrafi koordinatları ifade eden *enlem* ve *boylam* değerleri olsun,
- *x* ve *y* kartezyen koordinat sisteminde enlem ve boylama karşılık gelecek değerleri temsil etsin
- Dünyanın yarıçapı R olsun

Öncelikle bu değerlerin radyan cinsinden karşılıkları bulunur

- $enlem\_radyan = \pi * enlem / 180$
- $boylam\_radyan = \pi * boylam / 180$

bu değerler kullanılarak x ve y değerleri hesaplanabilir:

$$x = R * \cos(enlem\_radyan) * \cos(boylam\_radyan)$$
$$y = R * \cos(enlem\_radyan) * \sin(boylam\_radyan)$$

Ayrıca bu değerlerden geri enlem ve boylam değerleri hesaplamak için z değeri de bulunabilir:

$$z = R * \sin(enlem)$$

Dünyanın yarıçapı 6371km'dir.

Örneğin Elazığ üzerindeki enlem=38.65484, boylam=39.15383166666667 değerleriyle ifade edilen bir noktanın XY koordinat sistemindeki karşılığı x = 3858.083091684251, y = 3141.402500247374 olarak bulunmuştur.

x, y koordinatları verilen bir noktanın enlem ve boylam değerleri, radyan cinsinden aşağıdaki formüllerle bulunabilir:

enlem_radyan = asin (z / R)

boylam_radyan = atan2(y, x)

Bu değerleri coğrafi koordinatlara dönüştürmek için

*enlem = (enlem_radyan * 180) / π*

*boylam = (boylam_radyan * 180) / π*

formülleri kullanılır.

Bu formüller kullanılarak, verilen bir enlem-boylam çiftinin x-y karşılığını bulan aşağıdaki program yazılmıştır:

```
public class CoordConverter {

    //yeryüzünün yarıçapı
    private static double earthRadius=6371;

    public static double[] convertLatLon(double lat, double lon){
      double radlat = getRadian(lat);
      double radlon = getRadian(lon);
      double x = earthRadius *  Math.cos(radlat) * Math.cos(radlon);
      double y = earthRadius * Math.cos(radlat) * Math.sin(radlon);
      double [] ret = {x,y};
      return ret;
    }

    public static double getRadian(double a){
        double rad = Math.PI * a / 180;
        return rad;
    }

    public static void main(String[] args) {
        CoordConverter c = new CoordConverter();
        c.convertLatLon(40.7143528,-74.0059731);
    }
}
```

Bu program ile öncelikle GPS log dosyalarından elde edilen değerler XY koordinat sistemine çevrilmiş ve bu değerler kullanılarak da aşağıda anlatılan kümeleme çalışmaları gerçekleştirilmiştir.

4.7. GPS Veri Seti

Mekansal kümeleme testleri için, Naviskop (www.naviskop.com) araç takip sistemimdeki araçların konumunu barındıran log dosyalarından çıkarılan konum verileri kullanılmıştır.

Log dosyalarında veriler aşağıdaki gibi kaydedilmektedir.

```
T89*23DMN001,01,N3840.3771,E03911.3552,9,3467,0,188,180815,112027,H1075,LCC4C
#99*34FRNT002,001:A0:R0:S0,N4030.0694,E02918.9491,29,9018,101,183,180815,112028,H110
T99*34FRNT002,001:A0:R0:S0,N4030.0520,E02918.9448,18,9018,33,192,180815,112031,H110
#98*23TST23,001:A1925:R0:S0,N4042.7388,E03020.2315,81,20869,205,93,180815,112033,H
```

Bu satırlarda araçlara monte edilmiş GPS cihazlarından gelen, aracın plakası, ID'si, enlem, boylam, hız, yükseklik, saat, tarih, kontak durumu vb birçok farklı bilgi bulunmaktadır. Log dosyaları bir parser programı ile işlenip bu bilgilerin çıkarılması sağlanır. Parser programından elde edilen dosyada aşağıda gösterilen bilgiler bulunur:

```
73;084106;160615;false;40.7604;29.8149;4186.9374;2399.3367
63;084121;160615;true;38.6060;39.5196;3840.5584;3168.1239
63;084127;160615;true;38.6060;39.5200;3840.5338;3168.1531
```

Bu dosyada sırasıyla araç no, saat, tarih, kontak durumu, enlem, boylam, x koordinatı, y koordinatı bilgileri yer almaktadır. X ve Y koordinatlarının hesaplanmasında önceki bölümde anlatılan Kartezyen koordinat dönüştürme yöntemi kullanılmıştır.

Naviskop sisteminden alınan çok sayıda log dosyasından, 24-03-2015 tarihi ile 16-06-2015 tarihi arasında 598.143 adet nokta, 16-06-2015 tarihi ile 24-08-2015 tarihi arasında 892.999 adet nokta olmak üzere toplam 1.491.142 adet nokta elde edilmiştir.

Elde edilen bu veriler temizlendikten sonra ilk olarak Postgresql veritabanına atılmıştır. Burada yazılan sorgularla kümeleme testleri için veri seçimi gerçekleştirilmiştir. Veri temizliğinden sonra Postgresql veritabanında toplam 1.474.786 adet konum verisi kalmıştır.

Test verilerinde toplam 60 adet tekil araç tanımlıdır. Bu araçlardan 14 tanesi Türkiye'nin (30. Boylamın) batısında, İstanbul-Gebze bölgesinde hareket etmiştir. Geri kalan araçların önemli bir kısmının Elazığ ve çevresinde konum gönderdikleri görülmektedir. Dolayısıyla tüm araçları kapsayan bir kümeleme işleminde 2 büyük kümenin olması beklenebilir.

4.8. Hadoop Kümesi Üzerinde Apache Mahout ile Mekansal Veri Kümeleme

Bu bölümde, Google Cloud ve OpenStack kurulumu üzerinde kurulan Apache Hadoop kümeleri üzerinde, GPS verileri kullanılarak mekânsal veri kümeleme için takip edilen adımlar anlatılmıştır.

Mekansal verilerin Apache Mahout tarafından işlenebilmesi için SequenceFile denilen bir vektör veri formatına dönüştürülmesi gerekmektedir. Bu amaçla aşağıdaki program kullanılmıştır:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;
import org.apache.mahout.math.DenseVector;
import org.apache.mahout.math.VectorWritable;

class SequenceDosyaYap {

    private SequenceDosyaYap() {
    }

    public static final int SUTUN_SAYISI = 2;

    public void seqDosya(String inDosyaAdresi, String outdosyaAdresi, int maxSatirSayisi)
throws IOException {
        try {
            int DOSYA_SATIR_SAYISI = maxSatirSayisi;
            List<VectorWritable> liste = new ArrayList<>();

            BufferedReader br = null;
            br = new BufferedReader(new FileReader(inDosyaAdresi));
            String sCurrentLine;
            while ((sCurrentLine = br.readLine()) != null) {
                double[] sutunlar = new double[SUTUN_SAYISI - 1];
                for (int indx = 1; indx < SUTUN_SAYISI; ++indx) {
                    sutunlar[indx - 1] =
Double.parseDouble(sCurrentLine.split(",")[indx]);
                }
                liste.add(new VectorWritable(new DenseVector(sutunlar)));
            }
            Configuration conf = new Configuration();
            FileSystem fs = FileSystem.get(conf);
            Path path = new Path(outdosyaAdresi);
            String yol = path.toString();

            int file_no = 0;
            path = new Path(yol + "_" + file_no++);
            SequenceFile.Writer writer = new SequenceFile.Writer(fs, conf, path,
Text.class, VectorWritable.class);
            VectorWritable vec = new VectorWritable();
            int i = 0;

            for (VectorWritable vw : liste) {
```

```
                //dosyaya satırı yaz
                writer.append(new Text("ornek_" + i++), vw);
                //Her aaa sayıda yeni bir dosya oluştur
                if (i % DOSYA_SATIR_SAYISI == 0) {
                    writer.close();
                    i = 0;
                    path = new Path(yol + "_" + file_no++);
                    writer = new SequenceFile.Writer(fs, conf, path, Text.class,
VectorWritable.class);
                    vec = new VectorWritable();
                }
            }
            writer.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) throws Exception {
        String GIRIS_DOSYA = "gps.csv";
        String SONUC_DOSYA = "gps_bin_";
        int dosyaSatirSayisi = 1000;

        SequenceDosyaYap v = new SequenceDosyaYap();

        if (args.length == 0) {
            v.seqDosya(GIRIS_DOSYA, SONUC_DOSYA, dosyaSatirSayisi);
        } else if (args.length == 3) {
            v.seqDosya(args[0], args[1], Integer.parseInt(args[2]));
        }
    }
}
```

Bu program 3 adet giriş parametresi almaktadır. Bunlar

args[0] = XY konum (GPS) verilerinin olduğu dosya adı

args[1] = vektör çıkış dosyalarının adı

args[2] = çıkış dosyalarının satır sayısı

Örneğin 10.000 satırdan oluşan (10.000 adet x,y çifti barındıran) bir dosya 1000 satır taşıyan 10 adet vektör dosyasına bölünebilir.

Programda çıkış dosyalarının satır sayısının belirtilmemesinin temel sebebi çok sayıda koordinatın olduğu dosyaların daha küçük boyutlu vektör dosyalarına bölünmesidir. Çünkü büyük boyutlu dosyalar Hadoop üzerinde işlenememektedir. Bunun yerine veriler daha küçük boyutlu çok sayıda dosyaya bölünerek HDFS üzerine atılmaktadır.

Yukarıda verilen program komut satırından aşağıdaki gibi çağrılır. İçinde 1000 adet satır bulunan xy_bin.csv dosyası, içinde 100 adet satır bulunan 10 adet vektör dosyasına bölünsün:

```
java -classpath .:lib/* SequenceDosyaYap xy_bin.csv yuz 100
```

Bu komut çalıştırıldıktan sonra aşağıdaki şekilde isimlendirilen dosyalar oluşturulacaktır:

```
input/dist/yuz$ ls
yuz_0 yuz_1 yuz_10 yuz_2 yuz_3 yuz_4 yuz_5 yuz_6 yuz_7 yuz_8 yuz_9
```

Bu program ile istenildiği kadar büyük koordinat dosyalarının belirli büyüklükte vektör dosyalarına bölünmesi sağlanmıştır. Böylece HDFS'e çok büyük dosya atılmasına gerek kalmaz ve Hadoop işlemlerinde yetersiz hafıza hatası alınmaz.

4.9. Kümeleme Test Sonuçları

Bu bölümde, yukarıda detayları verilen veri setinden elde edilen çeşitli test verisi setleri ile, Google Cloud üzerinde kurulan HADOOP kümesi üzerinde yapılan MAHOUT K-Means testleri anlatılmıştır.

4.9.1. Hareketsiz Araçların Kümelenmesi

Araç konumlarının şehir ölçeğinde, araçların park halinde iken gönderdikleri konumlar kullanılabilir. Bunun için veritabanı tablosunda kontak bilgisinin false olduğu konumlar alınmıştır. Veritabanında yapılan sorguda araç kontak durumunun false olduğu toplam 529.986 adet konum bilgisi bulunduğu görülmüştür. Bunlardan 118.096 tanesi İstanbul bölgesinde, 411.890 tanesi ise İstanbul'un doğusundadır.

Bu testte öncelikle Türkiye çapındaki iki kümenin (İstanbul bölgesi, Elazığ bölgesi) bulunmasına çalışılmıştır. Bunun için Mahout k-means algoritması aşağıdaki komutla çalıştırılmıştır:

```
bin/mahout kmeans -i duran -c /cluster -o /seqfiles-final-cluster -x 1000 -k 2 -
ow --clustering -cd 0.5
```

Toplam 529.986 noktanın bulunduğu test dosyasının kümeleme işlemi 0.34 dakika sürmüştür. İşlem bittikten sonra kümeleme sonucu aşağıdaki komutla bir txt dosyasına aktarılmıştır:

```
bin/mahout clusterdump -i /seqfiles-final-cluster/clusters-4-final -p
/seqfiles-final-cluster/clusteredPoints -o duran.txt
```

Oluşturulan sonuç dosyası incelendiğinde iki kümenin yer aldığı, birinci kümede 395.028 nokta, ikinci kümede ise 134.958 nokta olduğu görülmüştür.

4.9.2. Tüm Verilerin Kümelenmesi

Elimizdeki test veri setinde bulunan toplam 1.474.786 adet noktanın Mahout ile k-means algoritmasına göre kümeleme testleri yapılmıştır. Bu testte iki ana kümenin (İstanbul bölgesi, Elazığ bölgesi) yanında bu bölgelerin arasındaki bölgelerde bir kümenin daha oluşacağı varsayımına göre 3 küme olacağı kabul edilmiştir.

Yaklaşık 1.5 milyon noktanın olduğu bu veri seti 500 bin adet nokta barındıran üç dosyaya bölünerek HDFS'ye atılmıştır.

```
java -cp .:lib/* SequenceDosyaYap tum-veriler.csv tum-veriler 500000
hadoop fs -mkdir tum-veriler-500bin
hadoop fs -put tum-veriler-500bin/* tum-veriler-500bin
```

Buna göre k-means algoritması aşağıdaki gibi çağrılmıştır.

```
bin/mahout kmeans -i tum-veriler-500bin -c /cluster -o /seqfiles-final-cluster -x
1000 -k 3 -ow --clustering -cd 0.5
```

Mahout ile kümeleme işlemi 0.79621 dakikada tamamlanmıştır. Clusterdump aracı ile kümeleme sonuçları bir txt dosyasına alınmış ve analiz edilmiştir.

```
bin/mahout clusterdump -i /seqfiles-final-cluster/clusters-6-final -p /seqfiles-
final-cluster/clusteredPoints -o tum-veriler.txt
```

4.9.3. Büyük Mekansal Verilerin Kümelenmesi ve Performans Testleri

Büyük veri teknolojilerinin avantajları, çok büyük miktarda verinin analiz edilmesi gerektiğinde ortaya çıkmaktadır. Yukarıda anlatılan testlerden de görüleceği üzere aslında karmaşık bir işlem olan kümeleme işleminin 1.5 milyon noktaya uygulanması bile sadece yaklaşık olarak 0.8 dakika sürdüğü görülmektedir.

Ancak bu rakamlar, veri sayısının çok çok büyük olduğu durumlarda sistem performansının nasıl olacağı konusunda bir fikir vermezler. Dolayısıyla tez çalışmasının son bölümünde Apache Mahout kullanarak, k-means kümeleme algoritmasının çok büyük

sayıda noktaya uygulanması gerçekleştirilmiş ve bu durumlar için sistemin performans analizi yapılmıştır.

Daha büyük sayılarda nokta elde etmek için elimizdeki 1.5 milyon nokta kopyalanarak çoğaltılmıştır.

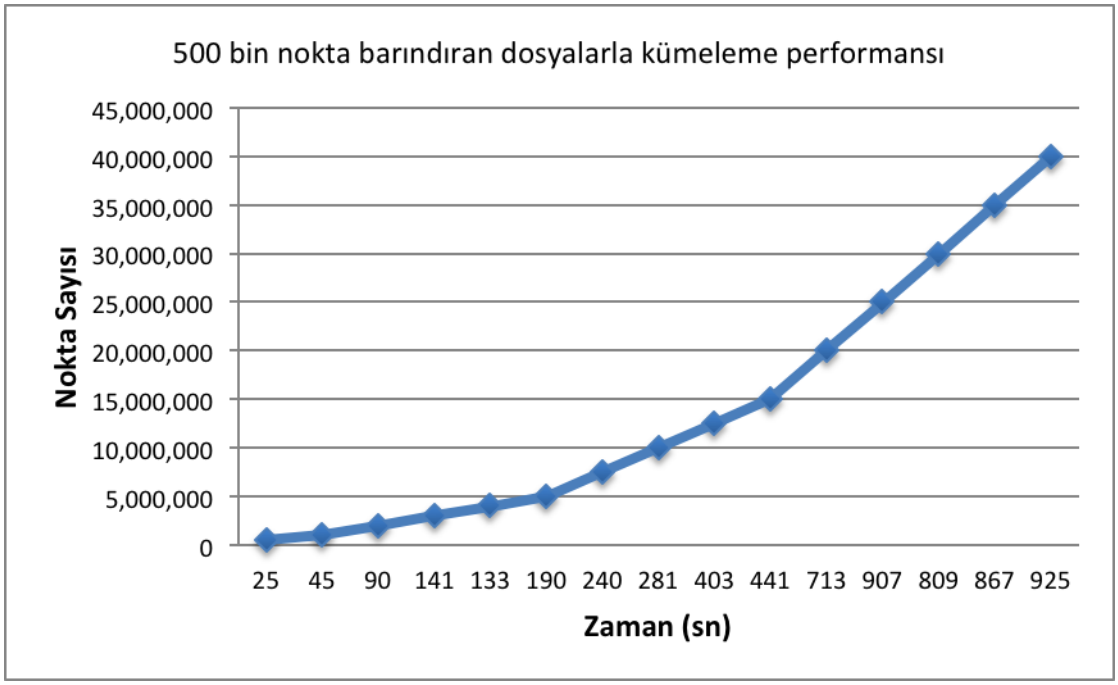
Bu bölümde ayrıca verilerin atıldığı HDFS dosya büyüklüğünün genel kümeleme performansına etkisi de incelenemiştir.

4.9.4. 500 Bin Nokta Barındıran Dosyalar Kullanılarak Yapılan Kümeleme Testleri

Bu testlerde her birisinde 500 bin adet enlem boylam bulunduran dosyalar kullanılarak aşağıda verilen tablodaki sayılara ulaşacak kadar noktanın kümelemesi işlemi gerçekleştirilmiştir.

Tablo 4.1. 500 bin adet nokta barındıran dosyalarla kümeleme süreleri

Nokta sayısı	Dosya Sayısı	Kümeleme süresi (ms)
500k	1	25.154
1m	2	44.632
2m	4	90.157
3m	6	141.241
4m	8	132.925
5m	10	189.761
7.5m	15	239.766
10.m	20	280.664
12.5m	25	403.311
15m	30	441.396
20m	40	713.368
25m	50	907.099
30m	60	808.906
35m	70	866.694
40m	80	925.110



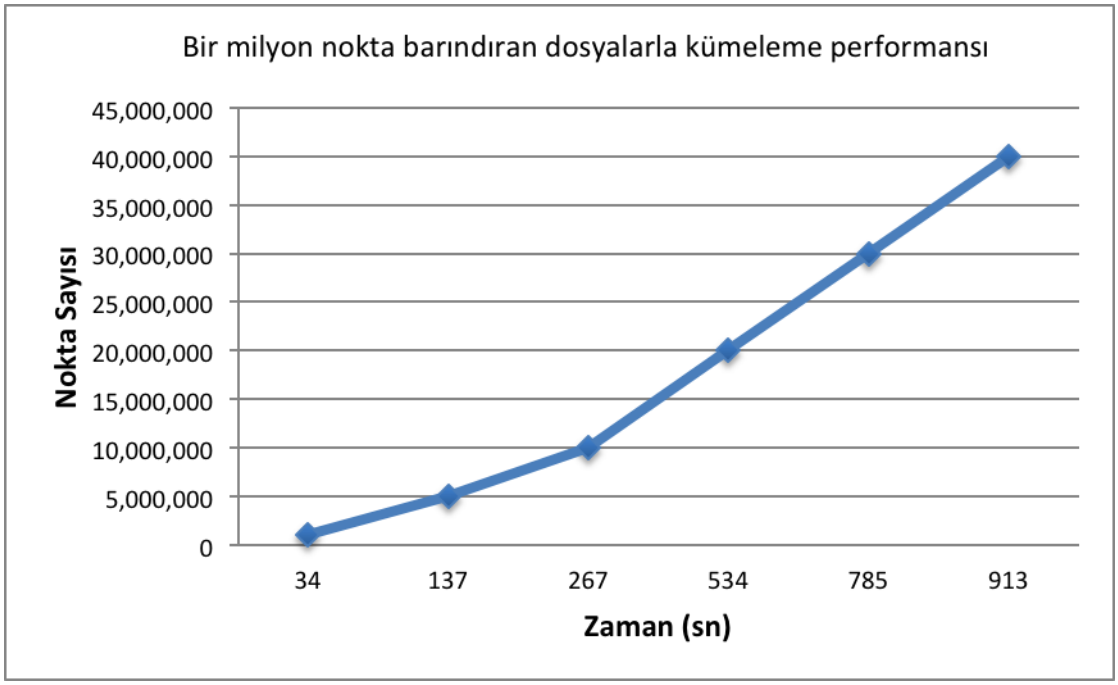
Şekil 4.24. Kümeleme zaman grafiği

4.9.5. 1 Milyon Nokta Barındıran Dosyalar Kullanılarak Yapılan Kümeleme Testleri

Bu testlerde her birisinde 1 milyon nokta bulunduran dosyalar kullanılmıştır.

Tablo 4.2. 1 milyon adet nokta barındıran dosyalarla kümeleme süreleri

Nokta sayısı	Dosya Sayısı	Kümeleme süresi (ms)
1m	1	33.598
5m	5	136.955
10m	10	267.105
20m	20	533.524
30m	30	784.720
40m	40	912.538



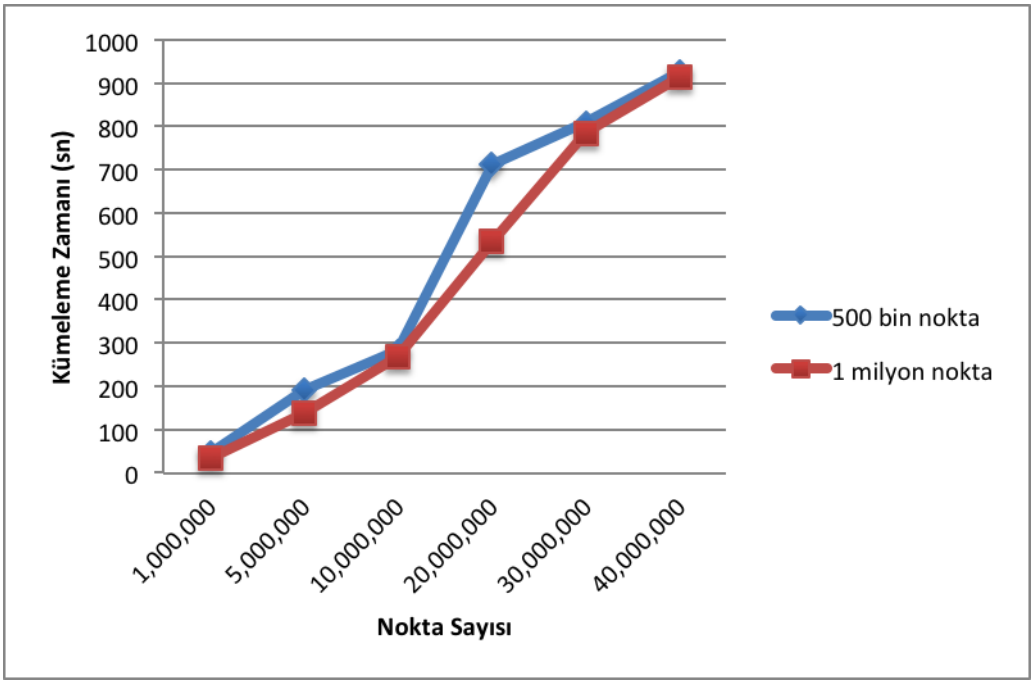
Şekil 4.25. Kümeleme zaman grafiği

4.9.6. Dosyalardaki Nokta Sayısının Kümeleme Performansına Etkisi

Yukarıda verilen iki farklı test sonuçlarının karşılaştırılması amacıyla, her iki tür testte aynı sayıda noktanın kümelenmesi için geçen süreler alınarak aşağıdaki tablo oluşturulmuştur. Birinci sütunda içinde 500 bin nokta barındıran dosyalarla elde edilen değerler, ikinci sütünda da her birisinin içerisinde 1 milyon nokta bulunan dosyalarla yapılan testlerin sonuçları verilmiştir.

Tablo 4.3. Kümeleme süreleri karşılaştırma tablosu

Nokta sayısı	Kümeleme süresi (ms) 500 bin noktalı dosyalar	Kümeleme süresi (ms) 1 milyon noktalı dosyalar
1m	44.632	33.598
5m	189.761	136.955
10m	280.664	267.105
20m	713.368	533.524
30m	808.906	784.720
40m	925.110	912.538



Şekil 4.25. Kümeleme performans karşılaştırması

Grafikten de görüldüğü gibi genel olarak, içerisinde 1milyon nokta barındıran dosyalarla yapılan testlerin daha kısa sürede tamamlandığı gözlemlenmiştir. Bunun temel sebebinin I/O için harcanan zamanın 500 binlik dosya sayısının 1 milyonluk dosya sayısının 2 katı olmasıdır.

5. SONUÇ

Web tabanlı harita hizmetlerinin, GPS alıcılarının ve konum bilgisi sunabilen mobil cihazların yaygınlaşması ile beraber mekânsal veri miktarında geçmişte görülmemiş oranda büyük artışlar meydana gelmiştir. Çok farklı türden uygulamalar tarafından kullanılma potansiyeline sahip bu verilerin işlenmesi, içerisinden anlamlı bilgilerin çıkarılabilmesi ve faydaya dönüşebilmesi için analiz edilmeleri gerekir. Ancak gelenekesel veri analizi yöntemleri çok büyük verilerin analizi için yetersiz kalmaktadır. Bu tez çalışmasının temel motivasyonu son yılların popüler konularından olan Büyük Veri teknolojilerinin mekânsal verilere uygulanabilirliğini araştırmaktır.

Tezin ilk bölümlerinde ilgili konular ve kullanılan teknolojiler açıklanmış, son bölümlerde ise tez çalışmasında kullanılan programların kurulum adımları gösterilmiş ve yapılan testler anlatılmıştır.

Tezde önemli bir Veri Madenciliği konusu olan kümelemeye odaklanılmış ve önemli kümeleme tekniklerinden birisi olan k-means algoritması kullanılmıştır.

Tezde Büyük Veri analizi için en çok bilinen sistem olan Apache Hadoop kullanılmıştır. K-means kümeleme algoritmasını Hadoop kümeleri üzerinde çalıştırabilmek amacıyla Apache Mahout adı verilen Dağıtık Makine Öğrenmesi kütüphanesi kullanılmıştır.

Çok sayıda mekânsal noktanın kümelemesinde Hadoop kümesi (cluster) oluşturabilmek için iki farklı yöntem denenmiştir. Öncelikle bulut.firat.edu.tr adresinde çalışan bir sunucuya OpenStack açık kaynak Bulut Bilişim altyapı yazılımı kurulmuş ve başlatılan sanal makinelerle bir küme oluşturulmuştur. Bu küme üzerinde de Hadoop ve Mahout kurularak testler yapılmıştır.

İkinci olarak da en büyük ticari Bulut hizmeti sağlayıcılarından birisi olan Google Cloud üzerinde kurulan Hadoop ve Mahout kümesi ile testler yapılmıştır.

Geleneksel Veri Madenciliği yöntemleri ve yaygın olarak kullanılan programlarla gerçekleştirilemeyecek kadar büyük kümeleme işlemlerinin dağıtık olarak yapılabildiğini göstermek amacıyla milyonlarca nokta barındıran veri setleri ile kümeleme işlemleri gerçekleştirilmiş ve bu işlemlerin aldığı süreler grafiklerle verilmiştir.

KAYNAKLAR

[1] **H. R. A. D. Luper, D. Cameron, J. Miller,** "Spatial and Temporal Target Association through Semantic Analysis and GPS Data Mining," in *IKE, 2007*, 2007, pp. 25–28.

[2] **R. N. Murty, G. Mainland, I. Rose, A. R. Chowdhury, A. Gosain, J. Bers, and M. Welsh,** "CitySense: An Urban-Scale Wireless Sensor Network and Testbed," in *2008 IEEE Conference on Technologies for Homeland Security*, 2008, pp. 583–588.

[3] **C. Zhou, D. Frankowski, P. Ludford, S. Shekhar, and L. Terveen,** "Discovering personally meaningful places," *ACM Transactions on Information Systems*, vol. 25, no. 3. p. 12–es, 2007.

[4] **K. Joseph, C. H. Tan, and K. M. Carley,** "Beyond 'local', 'categories' and 'friends': clustering foursquare users with latent 'topics,'" in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing - UbiComp '12*, 2012, pp. 919–926.

[5] **M. Kulldorff and N. Nagarwalla,** "Spatial disease clusters: detection and inference.," *Stat. Med.*, vol. 14, no. 8, pp. 799–810, 1995.

[6] X. Cao, G. Cong, and C. S. Jensen, "Mining significant semantic locations from GPS data," *Proc. VLDB Endow.*, vol. 3, no. 1–2, pp. 1009–1020, 2010.

[7] **S. S.Mohapatra, P. K. . Bhuyan, and K. V. . Krishna Rao,** "Genetic algorithm fuzzy clustering using GPS data for defining level of service criteria of Urban streets," *Eur. Transp. - Trasp. Eur.*, no. 52, 2012.

[8] **D. G. Jitesh Tripathi**, "Algorithm for Detection of Hot Spots of Traffic through Analysis of GPS Data," *J. IEEE Beac.*, no. January, p. pp 09–15, 2010.

[9] **Y. Zheng, L. Liu, L. Wang, and X. Xie,** "Learning transportation mode from raw gps data for geographic applications on the web," in *Proceeding of the 17th International Conference on World Wide Web '08*, 2008, pp. 247–256.

[10] **Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma,** "Understanding mobility based on GPS data," in *Proceedings of the 10th international conference on Ubiquitous computing - UbiComp '08*, 2008, no. 49, p. 312.

[11] **V. W. Zheng, Y. Zheng, X. Xie, and Q. Yang,** "Collaborative location and activity recommendations with GPS history data," *Proc. 19th Int. Conf. World wide web WWW 10*, p. 1029, 2010.

[12] **D. Ashbrook and T. Starner,** "Using GPS to learn significant locations and predict movement across multiple users," *Pers. Ubiquitous Comput.*, vol. 7, no. 5, pp. 275–286, 2003.

[13] **L. Jiancai and H. Xiao,** "A novel clustering algorithm based on gps of the mobile ad hoc network," in *Wireless Communications, Networking and Mobile Computing, 2009. WiCom'09. 5th International Conference on*, 2009, pp. 1–4.

[14] **Y. Zheng and X. Xie,** "Learning travel recommendations from user-generated GPS traces," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 1. pp. 1–29, 2011.

[15] "PVM (Parallel Virtual Machine)," 2015. .

[16] **P. S. Pacheco,** "Parallel Programming with MPI," *Perform. Comput.*, pp. 1–43, 1997.

[17] **R. L. Graham, T. S. Woodall, and J. M. Squyres,** "Open MPI: A Flexible High Performance MPI," in *Proceedings 6th Annual International Conference on Parallel Processing and Applied Mathematics*, 2005, pp. 228–239.

[18] "The Message Passing Interface (MPI) standard." [Online]. Available: http://www.mcs.anl.gov/research/projects/mpi/.

[19] **A. S. Tanenbaum and M. Van Steen**, *Distributed Systems: Principles and Paradigms, 2/E*. 2007.

[20] **J. Wu,** *Distributed system design*. CRC press, 1998.

[21] **G. Juve, E. Deelman, K. Vahi, G. Mehta, B. P. Berman, B. Berriman, and P. Maechling,** "Scientific workflow applications on amazon EC2," in *e-science 2009 - Proceedings of the 2009 5th IEEE International Conference on e-Science Workshops*, 2009, pp. 59–66.

[22] **P. Mell and T. Grance**, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," *Nist Spec. Publ.*, vol. 145, p. 7, 2011.

[23] **O. Şanlı,** "Bulut Bilişim," in *Akademik Bilişim*, 2011.

[24] **C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good,** "On the use of cloud computing for scientific workflows," in *Proceedings - 4th IEEE International Conference on eScience, eScience 2008*, 2008, pp. 640–645.

[25] **P. New and S. Cloud,** "IBM in the Cloud," *Int. J. Microgr. Opt. Technol.*, vol. 28, no. 4/5, pp. 4–7, 2010.

[26] **D. Robinson,** *Amazon Web Services Made Simple*. 2008.

[27] **E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good,** "The cost of doing science on the cloud: The montage example," in *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2008*, 2008.

[28] **T. Gunarathne, T. L. Wu, J. Qiu, and G. Fox,** "MapReduce in the clouds for science," in *Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010*, 2010, pp. 565–572.

[29] **K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa,** "Science clouds: Early experiences in cloud computing for scientific applications," *Cloud Comput. Appl.*, vol. 2008, pp. 825–830, 2008.

[30] **S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema,** "A performance analysis of EC2 cloud computing services for scientific computing," in *Cloud computing*, Springer, 2010, pp. 115–131.

[31] **P. Watson, P. Watson, P. Lord, P. Lord, F. Gibson, F. Gibson, P. Periorellis, P. Periorellis, G. Pitsilis, and G. Pitsilis,** "Cloud Computing for e-Science with CARMEN," *Newcastle Univeristy Rep.*, pp. 1–5, 2008.

[32] **OpenStack.com,** "OpenStack Operations Guide," *OpenStack.com*, no. OpenStack, 2013.

[33] **P. Sempolinski and D. Thain,** "A comparison and critique of Eucalyptus, OpenNebula and Nimbus," in *Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010*, 2010, pp. 417–426.

[34] **D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov,** "The eucalyptus open-source cloud-computing system," in *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID 2009*, 2009, pp. 124–131.

[35] **M. Hilbert and P. López,** "The world's technological capacity to store, communicate, and compute information.," *Science*, vol. 332, no. 6025, pp. 60–65, 2011.

[36] **M. Schroeck, R. Shockley, J. Smart, D. Romero-Morales, and P. Tufano,** "Analytics: The real-world use of big data," 2012.

[37] **S. Ghemawat, H. Gobioff, and S.-T. Leung,** "The Google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. p. 29, 2003.

[38**]** **J. Dean and S. Ghemawat,** "MapReduce," *Communications of the ACM*, vol. 53, no. 1. p. 72, 2010.

[39] **F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber,** "Bigtable," *ACM Transactions on Computer Systems*, vol. 26, no. 2. pp. 1–26, 2008.

[40] **I. Demir and A. Sayar,** "Hadoop plugin for distributed and parallel image processing," in *2012 20th Signal Processing and Communications Applications Conference (SIU)*, 2012, pp. 1–4.

[41] **C. Sweeney and S. Arietta,** "HIPI : A Hadoop Image Processing Interface for Image-based MapReduce Tasks," *Image Process.*, p. 5, 2010.

[42] **J. Lin and C. Dyer,** "Data-Intensive Text Processing with MapReduce," *Synthesis Lectures on Human Language Technologies*, vol. 3, no. 1. pp. 1–177, 2010.

[43] **S. Seo, E. J. Yoon, J. Kim, S. Jin, J. S. Kim, and S. Maeng,** "HAMA: An efficient matrix computation with the MapReduce framework," in *Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010*, 2010, pp. 721–726.

[44] **J. Ekanayake, T. Gunarathne, and J. Qiu,** "Cloud technologies for bioinformatics applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, pp. 998–1011, 2011.

[45] **S. K. B Karasulu,** "Modern Dağıtık Dosya Sistemlerinin Yapısal Karşılaştırılması," in *Akademik Bilişim 2008*, pp. 601–610.

[46] **A. S. Uğurcan Ergün, Süleyman Eken,** "Güncel Dağıtık Dosya Sistemlerinin Karşılaştırmalı AnaliziUğurcan," in *6. Mühendislik ve Teknoloji Sempozyumu*, 2013, pp. 213–218.

[47] **T. White,** *Hadoop: The definitive guide*, vol. 54. 2012.

[48] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "GraphLab: A New Framework for Parallel Machine Learning," *26th Conf. Uncertain. Artif. Intell. (UAI 2010)*, pp. 8–11, 2010.

[49] **Gürcan Yavuz, Sevcan Aytekin, Muammer Akçay,** "Apache Hadoop Ve Dağıtık Sistemler Üzerindeki Rolü," *Dumlupınar Üniversitesi, Fen Bilim. Ensititüsü Derg.*, vol. 27, pp. 43–54, 2012.

[50] **K. Shvachko, H. Kuang, S. Radia, and R. Chansler,** "The Hadoop distributed file system," in 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010, 2010.

[51] **S. Akbulut**, (2006) Veri Madenciliği Teknikleri ile Bir Kozmetik Markanın Ayrılan Müşteri Analizi Ve Müşteri Segmentasyonu," Gazi Üniversitesi, 2006.

[52] **S. K. Ömer AKGÖBEK,** "Veri Madenciliği Teknikleri İle Veri Kümelerinden Bilgi Keşfi: Medikal Veri Madenciliği Uygulaması," *New World Sci. Acad.*

[53] **C. C. Bojarczuk, H. S. Lopes, and A. A. Freitas,** "Data mining with constrained-syntax genetic programming: applications to medical data sets," in Proceedings Intelligent Data Analysis in Medicine and Pharmacology (IDAMAP-2001), 2001.

[54] **H. Jiawei and M. Kamber,** "Data mining: concepts and techniques," *San Fr. CA, itd Morgan Kaufmann*, pp. 377–385, 2001.

[55] **J. B. Macqueen,** "Some methods of classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.

[56] **T. Pang-Ning, M. Steinbach, V. Kumar, and others,** "Introduction to data mining," in *Library of Congress*, 2006, p. 74.

[57] **Ş. G. Öğüdücü,** "Veri Madenciliği Demetleme Yöntemleri."

[58] **H. Shimodaira,** "Clustering and Visualisation of Data," 2015.

[59] **M. Demiralay and A. Y. Çamurcu,** "Cure, Agnes ve K-Means Algoritmalarındaki Kümeleme Yeteneklerinin Karsılastırılmas," *İstanbul Ticaret Üniversitesi Fen Bilim. Derg.*, vol. 2, no. 8, pp. 1–18, 2005.

[60] **B. Diri,** "Makine Öğrenmesi Ders Notları," *Makine Öğrenmesi Ders Notları*.

[61] **A. Çabuk,** *Coğrafi Bilgi Sistemleri*. Anadolu Üniversitesi Yayınları No:2246, 2013.

[62] **C. D. Morais,** "GIS Data Explored – Vector and Raster Data," *GIS Lounge*.

[63] K. G. Lab, "Coğrafi Bilgi Sistemi (CBS / GIS) Nedir?" .

[64] **H. U. Endar Kelleci, Serdar Ergen,** *konumsal veritabanı*. T.c. Anadolu üniversitesi yayını no: 2341 Açıköğretim Fakültesi Yayını No: 1338, 2011.

ÖZGEÇMİŞ

Yağmur KILIÇ
Yeşil Vadi Mah. 8.Sokak Asude Kent Sitesi
A Blok No: 15 KIRIKKALE-Yahşihan
0 (554) 760 69 09
E-Posta: yagmurkilic29@gmail.com

GÜNCEL DURUM:

Fırat Üniversitesi Bilgisayar Mühendisliği Bölümünde, Yrd.Doç.Dr. Galip AYDIN'ın danışmanlığında "Yüksek Hacimli Mekansal Verilerin Dağıtık Veri Madenciliği Algoritmaları ile Analizi" başlıklı Yüksek Lisans tez çalışması yürütmekteyim.

Çalışılan konular :
- Dağıtık Sistemler
- Bulut Bilişim
- Büyük Veri
- MapReduce
- Apache Hadoop
- Spatial Hadoop
- Apache Mahout
- Veri Madenciliği

EĞİTİM:

2012 – Devam Ediyor
Yüksek Lisans: Fırat Üni. Bilgisayar Müh., Kuramsal Temeller

2008 – 2012
Lisans: Fırat Üni. Bilgisayar Müh.

2004 – 2008
Lise: Kırıkkale Anadolu Lisesi

İŞ DENEYİMİ:

2012 – Devam
TÜRKİYE BELEDİYELER BİRLİĞİ
BELBİS Belediye Yönetim Sistemi
Yazılım Mühendisi

2012 – 2014
YONCA CBS BİLİŞİM SİSTEMLERİ BİLGİSAYAR HARİTA MÜH. İNŞ. SAN.TİC. LTD. ŞTİ., FIRAT TEKNOKENT, ELAZIĞ
Web Tabanlı Coğrafi Bilgi Sistemleri Geliştirilmesi

http://www.naviskop.com, Naviskop Araç Takip Sistemi projesi.
http://uygulama.servisizle.com, Öğrenci Servisleri Takip Sistemi
http://cbs.firat.edu.tr, Fırat Üniversitesi İmar Bilgi Sistemi

Bu projelerde PostgreSql veri tabanı yöneticisi ve Geoserver harita sunucusu sorumlusu olarak çalıştım, ayrıca JSF ile arayüz geliştirme, Web Servisleri, SMS

entegrasyonu gibi konularda görev yaptım.

DİLLER: Türkçe, İngilizce

YAZILIM - TEKNOLOJİ BİLGİSİ:
Java
Java Servlets, JSP, JSF, PrimeFaces
HTML, JavaScript, CSS, JSON
JQuery, AJAX, XML
Openlayers, GeoServer, GML, KML
PostgreSQL, MySQL
MapReduce, Apache Hadoop, Apache Mahout
Linux

STAJLAR:

- YONCA CBS BİLİŞİM SİSTEMLERİ BİLGİSAYAR HARİTA MÜH. İNŞ. SAN.TİC. LTD. ŞTİ., Fırat Teknokent, Elazığ
  Staj (2011 – YAZ)
- DAS DOKUMAN ARŞİVLEME ve YÖNETİM SİSTEMLERİ A.Ş., ODTÜ Teknokent
  Staj (2010 - YAZ)

BURSLAR: TUBİTAK 2210-C ÖNCELİKLİ ALANLARA YÖNELİK YÜKSEK LİSANS BURSU (2013-2014)

KİŞİSEL BİLGİLER:
Doğum tarihi ve yeri: 26.09.1990, Kırıkkale
Sürücü ehliyeti: Var

REFERANSLAR:
Yrd. Doç. Dr. Galip AYDIN
Fırat Üniversitesi, Bilgisayar Mühendisliği Bölümü
Bilgi İşlem Daire Başkanı
0533 761 39 40
gaydin@firat.edu.tr