T.C.
İSTANBUL AYDIN ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BÜYÜK VERİ ORTAMINDA İKİ AŞAMALI KÜMELEME İLE ROTALAMA

YÜKSEK LİSANS TEZİ

Mehmet Fatih YÜCE

(Y1413.010026)

Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Ali GÜNEŞ

Mart 2017

T.C.
İSTANBUL AYDIN ÜNİVERSİTESİ
FEN BİLİMLER ENSTİTÜSÜ MÜDÜRLÜĞÜ

Yüksek Lisans Tez Onay Belgesi

Enstitümüz Bilgisayar Mühendisliği Ana Bilim Dalı Bilgisayar Mühendisliği Tezli Yüksek Lisans Programı **Y1413.010026** numaralı öğrencisi **Mehmet Fatih YÜCE**'nın **"BÜYÜK VERİ ORTAMINDA İKİ AŞAMALI KÜMELEME İLE ROTALAMA"** adlı tez çalışması Enstitümüz Yönetim Kurulunun 31.01.2017 tarih ve 2017/03 sayılı kararıyla oluşturulan jüri tarafından oybirliği.. ile Tezli Yüksek Lisans tezi olarak kabul.....edilmiştir.

Öğretim Üyesi Adı Soyadı **İmzası**

Tez Savunma Tarihi : 10.03.2017

1)Tez Danışmanı: Prof. Dr. Ali GÜNEŞ .....

2) Jüri Üyesi : Yrd. Doç. Dr. Metin ZONTUL .....

3) Jüri Üyesi : Yrd. Doç. Dr. Tuğba ALTINTAŞ .....

Not: Öğrencinin Tez savunmasında **Başarılı** olması halinde bu form **imzalanacaktır**. Aksi halde geçersizdir.

YEMİN METNİ

Yüksek Lisans / Doktora tezi olarak sunduğum “BÜYÜK VERİ ORTAMINDA İKİ AŞAMALI KÜMELEME İLE ROTALAMA” adlı çalışmanın, tezin proje safhasından sonuçlanmasına kadarki bütün süreçlerde bilimsel ahlak ve geleneklere aykırı düşecek bir yardıma başvurulmaksızın yazıldığını ve yararlandığım eserlerin Bibliyografya ’da gösterilenlerden oluştuğunu, bunlara atıf yapılarak yararlanılmış olduğunu belirtir ve onurumla beyan ederim. (10/03/2017)

Mehmet Fatih YÜCE

ÖNSÖZ

Tezin gerçekleşmesinde yardımlarından dolayı hocam Profesör Doktor Ali Güneş'e ve katkılarından dolayı da enstitümüze teşekkürü bir borç bilirim.

Mart,2017 **Mehmet Fatih YÜCE**

İÇİNDEKILER

Sayfa

KISALTMALAR

TW	: Time Windows
ZA	: Zaman aralığı
CSV	: Virgül ile ayrılmış dosya yapısı
DFS	: Dağıtık Dosya Sistemi
HDP	: Hortonworks Hadoop Dağıtımı
CDH	: Cloudera Hadoop Dağıtımı
UI	: Kullanıcı Ara yüzü
Pio	: Prediction.IO
AÇT	: Afrika'dan Çıkış Teorisi
ARP	: Araç rotalama Problemi/
VRP	: Vehicle Routing Problem
TDP	: Truck Dispatching Problem
ADP	: Araç Değnekleme Problemi
NP	: Belirsiz Turing Makinesi ile çokterimli (polinomsal) zamanda çözülebilen karar problemleri
NP-hard	: NP-zor
GSP	: Gezici Satıcı Problemi
TSP	: Travelling Salesman Problem
PKP	: Paket Kutulama Problemi
BPP	: Bin Packing Problem
ÇBPKP	: Çok Boyutlu Paket Kutulama Problemi
MBPP	: Multidimensional Bin Packing Problem
M/R	: Map/Reduce (Haritala ve Azalt)
H/A	: Haritala ve Azalt
NDFS	: Nutch Dağıtık Dosya Sistemi (Nutch Distributed File System)

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

BÜYÜK VERİ ORTAMINDA İKİ AŞAMALI KÜMELEME İLE ROTALAMA

ÖZET

Bu tezde yeni bir fikir olarak düşünülen araç rotalamada zaman aralığı (time windows) kümeleme konusunda yeni bir yöntem ele alınacaktır. Daha önceki çalışmalarda nokta yönelimli kümeleme ile başarılı sonuçlar alındığı bilinmektedir. Bu çalışmada ise nokta yönelimli kümeleme ile zaman aralığı kümeleme aynı anda uygulanıp daha uygun bir çözüm bulunabileceği ele alınacaktır. Uygulanacak metot büyük veri üstünde uygulanarak bu gibi ihtiyaçları olan sistemlerde kullanılabilirliği anlatılacaktır.

Anahtar Kelimeler: *Büyük Veri, Yüklü ve Zaman Aralıklı Araç Rotalama*

TIME WINDOW AND LOCATION BASED CLUSTERED ROUTING WITH BIG AND DISTRIBUTED DATA

ABSTRACT

In this thesis, thought as a brand new idea, a vehicle routing algorithm will be presented. It will be based on time windows and location based clustering. Previous works on location based clustering are prooved to be successfull. In this thesis, location based clustering and timewindow based clustering is applied at the same time. It shown that it can generate good results. Proposed method is applied on a big data platform and also usability on such system is also described.

Keywords: *Big Data, Capacited Vehicle Routing With Time Windows*

1 GİRİŞ

Ulaşım bugün en büyük ihtiyaçlarımızdan birisidir. Yapı itibarı ile hareket kabiliyetine sahip olduğumuzdan dünya üzerinde, ister ticaret amacı ile ister başka amaçlar için olsun, gitmek istenilen yerlere ulaşım ilk varoluşumuzdan beri üstünde çalışılan ve geliştirmek için çeşitli teknolojiler geliştirilen bir alandır.

Yürümek doğuştan sahip olduğumuz bir özelliktir. Tarihin başlangıcında atalarımızın yürüyerek kat ettiği mesafeler inanılmaz boyutlardadır. "Afrika'dan Çıkış (AÇT)" (Nei 1995) teorisine göre (Out of Africa Theory), modern insan ilk olarak Etiyopya platosunda gelişmiş, coğrafi koşulların elverişliliği nedeni ile de iki rota kullanarak Ön Asya'ya, Avrupa kıtasına ve Hindistan alt kıtasına yayılmışlardır. Bu göç dalgası nesiller boyu sürmüş, kullanılan başka bir teknoloji olmadığı varsayılarak sadece yürüyerek başarıldığı düşünülmüştür. Bu yayılım kasıtlı ve programlı bir yürüyüş olmamakla beraber yaşamaya elverişli olan alanlara yerleşim ve nesiller sonra nüfus artışı ve kaynakların tükenişi gibi nedenler ile daha ilerilere göç şeklinde olmuştur.

Orta Asya'da yaşayan atalarımız ise daha etkili bir ulaşım aracı olan atları kullanmış ve daha kısa bir zamanda daha uzun bir alana yetişme imkânını elde etmiştir. Bu yeni teknoloji, Afrika'nın insanlığının beşiği olması gibi, Orta Asya'nın insanlığın gelişip serpildiği ve yayıldığı en büyük merkezlerden biri olmasını sağlamıştır (Bouckaert et.al. 2012). Böylece hareket kabiliyeti artan insanların büyük göç dalgaları halinde yayılma devirleri başlamıştır.

Tekerleğin icadı ise bambaşka ufuklar açmış, noktalar arası daha büyük miktarda yük ve insan taşınmasını sağlamıştır.

Göçler genellikle en kısa yollardan yapılmaya çalışılmıştır. Her ne kadar iki nokta arası en kısa mesafe düz bir çizgi olarak kabul edilse de (kuş uçuşu mesafesi), coğrafyanın izin verdiği kadarı ile bu başarılmaya çalışılmıştır. Örneğin İç Anadolu'dan Çukurova'ya en kısa mesafe dağların yol vermez

konumu nedeni ile biraz daha uzunca olan Gülek Boğazı üzerinden yapılabilmektedir.

Tarihte yapılan göçlerde zaman, ekonomi ve miktar zamanımızda yapılan ulaşımlarda olduğu kadar önemli olmamaktadır. Bugün bir taşımacılık yapılacaksa sorulan sorular şunlardır;

- En kısa mesafe
- En kısa sürede
- En çok yük
- En ucuz şekilde

nasıl taşınır şeklindedir. Duruma göre soruların hepsi veya bir kümesi sorulabilmektedir. Ulaşım imkânlarının çeşitliliği ve bunların ekonomisi, verilecek cevabın ne olacağını belirlemektedir. En kısa mesafe ile en kısa süre her zaman aynı anlama gelmemektedir. Mesafe kısa olabilir ama yeryüzü şekilleri, trafik ve araç yoğunluğu süreyi etkileyebilmektedir. Son madde (en ucuz şekilde) çoğunlukla diğer üç maddenin bir sonucu olmaktadır.

Son iki yüzyıl içerisinde ulaşım teknolojisinde meydana gelen gelişmeler kullanılabilecek yöntemlerin sayısını da arttırmıştır. Önceleri buharla çalışan daha sonra petrol temelli hale getirilen motorlar canlı kuvvetine dayanan yöntemlerin yerine geçmiş, ekonomi çok önemli bir kalem olmaya başlamıştır.

Geçen yüzyıl içerisinde yukarda sorduğumuz her bir sorunun cevabını bulmak amacı ile yoğun çalışmalar yapılmış, her bir soru başka bilimsel çalışma alanlarına ve problemlere ayrılmıştır. Her bir alanda bir veya birkaç sorunun cevabı bilimsel olarak ele alınmış, çözüm sezgisel veya optimal şekilde verilebilmiştir.

Taşımacılıkta kullanılan filo tipi de önem arz etmektedir. Homojen yapıda filolar olabildiği gibi heterojen yapıda filolarda bulunmaktadır. Bazı filolar aynı tip ve çeşitten oluşan araçlardan oluşabildiği gibi değişik şekil ve tiplerde de olabilirler. Örneğin tek tip ve tek çeşit kamyon kullanan filolar (homojen) olabileceği gibi, kullanılan tipteki araçların da farklı yapıda olabileceği filolar da bulunabilir (heterojen). Kullanılan araçların değişik miktarda yük taşıyabileceği kurulumlar da yapılabilmektedir. Araç tipi dendiğinde anlamamız gereken kaynak gurupları olmalıdır. Tır tipinde bir araçtan bahsedildiğinde, bu

tanımın içerisine çeşitli miktar ve ekonomilerde taşıma yapabilen çeşitte tırlar anlaşılmalıdır. Bazıları yüksek miktarda taşıma yapabilirken bazıları daha az miktarda taşıma kabiliyetine sahip olabilirler. Bu çalışmada, kullanılan filo homojen yapıdadır. Kullanılan araçlar tek tipte ve tek çeşittedirler, ayrıca ekonomi olarak da, araç tipi sabit kabul edilmiş ve hesaplamalara katılmamıştır.

Taşınan yükün birimi de önemli olmaktadır ve değer olarak da genellikle metrik sistem (metric system) kullanılmaktadır. Birim çok boyutlu bir yapı da olabilmektedir. Örneğin bir yükün birimi dendiğinde ağırlık, hacim ve palet miktarları hesaba katılmak zorunda olabilir. Bu durumda problem çok boyutlu paket kutulama problemini/ÇBPKP (Multidimensional Bin Packing Problem/MBPP) de içerebilir. Her ne kadar gerçek hayatta çok boyutlu birimler kullanılıyor olsa da, bu çalışmada tek boyutlu yüklerden bahsedilecektir. Birim denecek bu miktar, istenilen sistemde bir değer olarak anlaşılabilir. Bir koşmada hacim olarak düşünüleceği gibi, başka bir çalışmada ağırlık olarak da ele alınabilir.

1.1 Tezin Amacı

Son yirmi yılda teknolojide yapılan ilerlemeler ve taşınan miktarda meydana gelen artışlar klasik çözümlerde iyileştirme yapmaya gidilmesine neden olmuştur. Problem büyüdükçe de, en iyi sonuç bulma hedefinden vazgeçilmiştir. Bunun nedeni en iyi sonucu bulmak için yapılabilecek hesaplamaların şimdiki bilgisayar teknolojisi ile kabul edilemeyecek sürelerde yapılabilmesidir. En iyi çözüm birleştirme (combination) hesapları ile bulunabileceğinden, en iyi sonucu bulan algoritmalar (exact algorithms) yerine sezgisel algoritmalar geliştirilmiştir. Sezgisel algoritmalarla en iyi sonuç bulunamazsa bile yakın bir çözümün daha uygun bir zamanda bulunabilmesi sağlanmıştır. Bu çalışmada kullanılan yöntem de sezgisel bir yöntemdir.

Üstünde duracağımız problem değişken bir schedule sistemine sahip bir yapılanma için çözüm üretmek olacaktır. Planlama durağan değildir. Yani herhangi bir zamanda taşınması gereken yeni bir yük gelebilmektedir. Bu yük daha önce taşınması planlanan ve henüz yola çıkmamış veya yola çıktığı halde varacağı yerden bu yükü alabilecek araçlarla taşınabilecektir. Miktar olarak da bir sınır yoktur. Sistemin yatay olarak büyüyebilmesi sağlanmıştır. Örneğin bin

isteği planlamak için bir makine yeterken bu istekler iki bine çıkarsa iki adet makine ile birlikte çalışılarak aynı sürede planlamayı yapmaktadır. Bunun için de büyük veri sistemlerinden faydalanılmıştır.

Yeni bir yöntem olarak da zaman aralığı yönelimli kümeleme yöntemi ve nokta kümeleme yöntemlerinin birleştirilmesi ele alınacak, bununla daha iyi ve hızlı sonuç bulunabileceği gösterilecektir.

1.2 Literatür Taraması

Araç rotalama Problemi/ARP (Vehicle Routing Problem/VRP), veya Araç Değnekleme Problemi/ADP (Truck Dispatching Problem/TDP), çokterimli (polinomsal) zamanda çözülebilen karar problem zorluğunda (deterministic polynomial-time hard/NP-hard) bir problemdir ve Gezici Satıcı Problemi/GSP'nin (Travelling Salesman Problem/TSP) bir genelleştirilmesi şeklinde ele alınabilir. GSP'nin ne zaman geliştirildiği belli olmamasına rağmen (Hoffman, 1986), ilk olarak (Flood, 1955) tarafından akademik dünyaya tanıtıldığı düşünülmektedir.

Problem şu şekilde tanımlanmıştır;

- G: çıkış yapılan noktaya dönen bir yol (Hamiltonian path) veya bir bütün çizelge (complete graph)
- V: nokta veya konum kümesi,
- E: kenar kümesi,
- c_{ij}: E içerisinde tanımlanan her kenar için bir maliyet

verilmiştir. c_{ij} $k \in V$ 'den $m \in V$'e gitmek için tanımlanmış maliyettir (Zambito, 2006)

ARP ise ilk olarak (Dantzig G B, 1959) tarafından ele alınıp, şu şekilde tanımlanmıştır;

- R: rota kümesi
- V: bir araç filosu tarafından hizmet gören araç kümesi
- P: bir müşteri kümesini gezen bir araç,
- C: bir constraint kümesi

(Kumar, 2012). Böylece bir çözüm kümesi içerisinde bulunan her bir rota bir GSP problemine dönüşür.

ARP'ler yapısına veya modeline göre sınıflandırılabilir (Granada, 2016). Bu açıdan, aşağıdaki belirteceğimiz liste tamam olmamakla birlikte ARP sınıflarını belirtmektedir. Her birinin ayrıca (aksi belirtilmediği takdirde), yüklü veya zaman aralıklı halleri de ayrı bir problem kümesi olarak ele alınabilir.

Yüklü Araç Rotalama Problemi (Capacitated Vehicle Routing Problem/CVRP)

Bu problemde, geleneksel ARP ile birlikte, paket kutulama problemi/PKP (Bin Packing Problem/BPP) aynı problem içerisinde ele alınır (Ralphs, 2003). Araç aynı miktara (hacim, ağırlık veya palet) sahip yüklerden oluşmaz. Yükler heterojen yapıda olabilmektedir. Ayrıca aynı ağırlığa sahip yükler, hacim veya palet olarak farklı değerlere sahip olabilmektedir. Bu tezdeki amaçlarımızdan birisi bu problem için yeni bir yöntem sunmaktır.

Asimetrik Yüklü ARP (Asymmetric Capacitated VRP/ACVRP)

Bu kümede maliyetler gidiş ve geliş olarak ayrı olarak sınıflandırılmıştır (Vigo, 1996). Müşteriye giderken oluşan maliyet, aracın daha önce ilk yolculuğa çıkış yerine gelirken taşıdığı maliyetten farklıdır.

Yay Rotalama Problemi (Arc Routing Problem/ARP)

Araç rotalama problemine benzemekle birlikte, burada asıl amaç belirli kenarların tümünü gezebilme imkânını bulmaktır (Eiselt, 1995). Tek araç olduğu kabul edilir. Aracın kapasitesinin de sınırsız olduğu düşünülür.

Ulaşım İçin İstek Yapma Problemi (Dial-a-ride Problem/DARP)

Bu problemde noktalar ne zaman alınmak istediklerini, ayrıca ne zaman istedikleri yere bırakılmak istediklerini sisteme belirtirler (Cordeau, 2003). Amaç tek araçla mümkün olan en çok noktadan alınması gerekenleri almak ve en ucuza taşımadır. Alınma ve bırakılma zamanları genellikle bir nokta değildir. Örneğin bir kullanıcı beni saat iki ile üç arasında almalısınız diyebilmektedir.

Emisyon Azaltıcı ARP (The Emissions Vehicle Routing problem/EVRP)

Bu rotalama probleminde, diğer kısıtlarla birlikte emisyon miktarını düşürme ayrı bir amaçtır (Jemai, 2012, April).

Zaman Aralıklı ARP (Vehicle Routing Problem with Time Windows/VRPTW)

Geleneksel ARP'de araç ne zaman istenilen noktaya varırsa varsın problem teşkil etmemektedir. Bu problem ise aracın çıkış ve varış saatleri belirli zaman aralıkları arasında olmak zorundadır (Bräysy, 2005). Bu zaman aralıkları sabit olabileceği gibi değişken de olabilmektedir. Bu tezdeki amaçlarımızdan birisi bu problem için de yeni bir yöntem sunmaktır.

Genel ARP (Generalized VRP/GVRP)

Bu problemde amaç gidilecek yerlerin kısıtlanmasıdır (Baldacci, 2010). Yer rotalama probleminin özel bir durumu olarak da düşünülebilir (Kara, 2003). Rotalama yapılırken bir noktadan nerelere gidileceği daha önceden saptanmış veya iyileştirilmiş olabilmektedir. Bu nedenle problem daha küçük bir hale getirilmekte ve iyileştirme sırasında daha uygun bir rota daha uygun bir zamanda bulunabilmektedir.

Yer Rotalama Problemi (Location Routing Problem/LRP)

Yer rotalama probleminde araç rotalanması öncesi kurulacak depoların veya merkezlerin lokasyonu iyileştirilmeye çalışılmaktadır (Nagy, 2007). Daha sonra yapılacak rotalamalar bu merkezlerden başlayarak veya bu merkezlere giderek iyileştirilecektir (Savran A. I., 2015).

Uzaklık Kısıtlı ARP (Distance Constrained Vehicle Routing Problems/DCVRP)

Bu rotalama probleminde her hangi bir rota daha önce belirtilen bir değerden daha fazla olamamaktadır (Laporte, 1984). Bu kıtı her bir rotaya verilebildiği gibi, tüm rotaların toplamı şeklinde de verilmiş olabilir. Ayrıca her rotanın ve tüm çözümün de ayrı ayrı alabileceği uzunluk değerleri olabilmektedir.

İki-Parçalı Araç Rotalama Problemi (Two-Echelon Vehicle Routing Problem) (2E-VRP)

Bu ARP kümesinde (Gonzalez-Feliu, 2008) veya uygulamalarında (Crainic, 2010), yük bir çıkış noktasından müşteriye götürülürken, araç dolum oranını yükseltme ayrıca maliyeti düşürme amacı ile araya bir depo konur. Buna benzer şekilde çok parçalı bir yapıda olan hali de bulunmaktadır.

Yük bırakma ve Almalı ARP (Vehicle Routing Problem with Pickup and Delivery/VRPPD)

Çeşitli yapıları bulunmakla beraber, araçlar bir noktaya hem yük bırakabilmekle birlikte aynı noktadan da yük alıp başka noktalara bırakmak üzere rotalanabilmektedir (Çatay, 2010). Genellikle depolar arası taşımacılık için bir yöntem olarak düşünülür.

Vehicle Routing Problem with Backhauls (VRPB)

Yük bırakma ve Almalı ARP problemine benzemekle birlikte, bu problemde tüm yükler bırakılmadan yeni yük alınmaz (Crispim, 2005).

Çalışmada ele alacağımız problem ise yüklü ve ayrıca zaman aralıklı araç rotalama problemidir.

1.3 Benzer Çalışmalar

Bilinen bir sistem beta düzeyinde yaptığı bir metot ile zaman aralığı temelli rotalama için denemeler yapılmıştır. Ancak rota uzunluğu büyüdükçe sistemin tek bir rota oluşturması için gereken süre logaritmik olarak arttığı görülmüştür. Bu sistemde ilk noktadan başlayarak ulaşılabilecek noktalar bulunmaktadır ve bu noktalardan sonra varılabilecek noktalar da aynı şekilde bulunmaya çalışılmıştır. Kısa uzunluklu rotalar için bu sistem çözüm üretebilmiştir ancak rota uzunluğu beş ve üzeri olduğunda sistem tıkanmıştır (Sakalli, 2013).

Araç rotalamada zaman aralığı kümeleme yapan sistemler bu çalışma hazırlanırken bulunamamıştır. Ancak diğer bazı sistemlerin zaman aralığı kısıtı uyarlama yaptığı görülmüştür (Solomon, 254-265.). Rota oluştuktan sonra zaman aralığı uygunluğu düzenlenip, ardından belirlenen kısıtlara göre maliyeti hesaplama yapıp, bu maliyet daha önce bulunan rotalardan daha iyi bir değerde ise kabul etme şeklinde bir yöntemleri olduğu anlaşılmıştır.

Ayrı bir yöntem ise zaman aralığını parçalama veya bölmedir (Yildirim, 2012). Bu yöntemde ilk noktadan herhangi bir zaman değil de, belirli aralıklar içerisinde çıkış yapma şeklinde bir yöntem izlenmiştir. Örneğin 12 ile 18 arası olan bir aralığı, 12-15 ve 15-18 arası bölme ve zaman aralığını bu olarak kabul etme üzerine çalışılmıştır.

2 KÜMELEME NEDİR?

Veri miktarı arttıkça araç rotalama uygulamaları yavaşlamaya veya çok büyük veriler girdi olarak alınmaya başlandığında uzun süren süreçler ile cevap verebilmektedirler. Bunun önüne geçebilmek için ise önerilen yöntemlerden birisi kümelemedir (Beasley, 1983). Böylece girdi olarak ele alınacak veri kümesi küçültülüp birden çok rotalama problemi olarak ele alınmaktadır. Kümeleme işlemlerinin bir faydası yapılacak işlemlerin parallel olarak işleme sokulabilmesidir. Daha küçük kümelere üstünde yapılacak işlemler daha kısa zaman alacaktır. Böylece çözüm uzayı küçüldüğünden yapılabilecek işlemlerde hızlanmış olacaktır. Özellikle kombinasyon işlemleri içeren bir problem çözümünde bu yolla uygun çözümler bulabilmek çok daha kolaylaşacaktır. Kombinasyon işlemlerinde girdi veri miktarı belli bir miktardan sonra çözüm için gereken zaman mantık sınırları dışına çıkmaktadır. Zamanımızdaki bilgisayar teknolojisinin uygun bir süre içerisinde çözebileceği problemlerin dışına çıkmaktadır.

Kümeleme yöntemi olarak en çok kullanılan nokta yönelimli kümelemedir. Bu yöntemde coğrafi koordinatlar üstünden birbirine yakın bulunan noktalar ayni küme içerisine alınmakta ve rotalama işlemi bu kümelerde ayrı ayrı yapılmaktadır (Dondo, 2007). Her bir kümede rotalama işlemi bittiğinde ise, var olan araç miktarı ve durumuna göre, o anki durum çözüm olarak kabul edilebilir veya rotalama işlemleri bir yere geldiğinden tüm küme üstünden iyileştirme çalışmaları yapılabilir. Bu sonra çalışma bulunan çözüm üstünde sınırlarda bulunan atamaları değiştirerek bir iyileştirmeye neden olabilir. Bazen de sadece sınırdaki atamalar değil de daha içerilerde bazı değişikliklerle daha iyi bir çözüme neden olabilmektedir.

2.1 Kümeleme İşlemlerinin Faydaları

Yukarıda bahsedilen kümeleme işleminin birçok faydası olmakta ve iyileştirme işlemleri sırasında bolca kullanılmaktadır. Bu faydaları aşağıdaki gibi sıralamak mümkündür;

- Çözüm Uzayı Küçültme
- İşlem Hızlandırma
- Aynı anda işlemleyebilme

2.1.1 Çözüm Uzayı Küçültme

Her hangi bir algoritmik iyileştirme çözümünü ele alırsak alalım, girdi veri miktarı arttıkça algoritmanın çalışması uzayacaktır. Bunun nedenlerinden birisi, artan her bir veri miktarının hesaplama sırasında alacağı zamandır. Herhangi bir kombinasyon hesabına bakarsak ve her bir işlemin çalışması bir birim olarak düşünülürse, aşağıdaki gibi bir hesaplama mümkün olabilir. Kombinasyon formülü aşağıdaki gibidir;

$$C(\mathrm{n},\mathrm{r}) = (C_r^n) = \frac{n!}{r!\,(n-r)!}$$

Görüldüğü gibi, girdi eleman miktarı artıkça hesaplama miktarı çarpı yeni miktar ((n+1)*(ilk zaman)) olarak artmaktadır. Şu şekilde gösterebiliriz; Yukardaki formülü ilk n elemanın hesaplanması için geçen zaman olarak kabul edersek, n+1 eleman için yapmamız gereken hesaplama şu şekilde olur;

$$C(\mathrm{n}+1,\mathrm{r}) = (C_r^{n+1}) = \frac{(n+1)\,n!}{r!\,((n+1)-r)!}$$

Şekline dönüşür. Değer vererek yaparsak, n=10 ve r=4 kabul edersek (on adet noktanın her bir aracın 4 adet noktaya gidecek olarak rotalanması olarak kabul edilebilir), işlem süresi

$$C(10,4) = (C_4^{10}) = \frac{10!}{4!\,(10-4)!} = \frac{3628800}{24(6)!} = \frac{3628800}{17280} = 210$$

birim zaman alacaktır. Girdi veri miktarını arttırmak istersek ve n=20 ve r=4 dersek;

$$C(20,4) = (C_4^{20}) = \frac{20!}{4!\,(20-4)!} = \frac{2432902008176640000}{24(16)!}$$
$$= \frac{2432902008176640000}{502146957312000} = 4845$$

olur. Girdi veri miktarının iki kat artması, hesaplama zamanında yaklaşık yirmi üç (23) kat bir artış anlamına gelmektedir. Eğer iki kat girdi veri yerine kümeleme yapmış olsa idik 210+210=420 bir zamanda işlemi bitirebilecektik. Yani iki adet küme kendi içerisinde onun dörtlü kombinasyonları şeklinde hesaplanırsa, işlem zamanı sadece girdi veri miktarı kadar artar. Bire bir olabilecek bir artıştan bahsetmiş oluruz.

2.1.2 İşlem Hızlandırma

Bir önceki bölümde bahsedilen gibi, çözüm uzayı küçüldükçe işlem hızlandırma kendiliğinden olacaktır. Çünkü rotalama sırasında her bir çözüm üçün ayrı bir hesaplama yapmak gerekir. Bir rota adayı düşünüldüğünde üstünde yapılacak işlemler bellidir.

1. Noktalar arası uzaklıkları bulma ve rota uzunluğunu bulma
2. Başlama zamanını ayarlama ve her bir durağa varış ve her bir duraktan ayrılış zamanlarını bulabilme.
3. Zamanlamalar tamamlandıktan sonra bu zamanların belirli bazı kurallara uygunluğunun bulunması (tatil günleri vs.).
4. Zamanla uygun değilse zamanlamanın düzenlenmeye çalışılması.
    a. Aracın daha erken veya daha geç sefere çıkarılması
    b. Her bir durakta bekleme sürelerinin hesaba katılarak uygun bir zamanla hesap edilebilmesi
    c. Zamanlama ayarlanamadı ise son çare olarak bazı durakların seferden çıkarılarak seferin zamanlama olarak uygun hale getirilmeye çalışılması

Bunun gibi her bir rota adayı birçok işleme uğratılır. Bunlar ardından her bir rotanın belirli bir maliyeti çıkarılır. Bu noktalara uğrayan bir rota her zaman da uygun bir rota olmayabilir. Örneğin zamanla uygun bir şekilde ayarlanamadığında bu rotanın kabul edilmesi uygun olmamakta ve atılmaktadır. Atılan her bir rota yine hesaplama zamanından yemektedir.

Yapılan tüm işlemler girdi veri miktarı arttıkça daha uzun zaman almaktadır. Bir önceki bölümde belirtilen girdi veri miktarı iki kat arttığında hesaplama zamanı 23 kat artmaktadır. Ancak kümeleme yöntemi ile sadece iki kat bir artış meydana gelmektedir. Bir sonraki bölümde bu iki kat artılın bile destekleyen donanım üzerinde eski hızda nasıl hesaplanabileceği anlatılacaktır.

2.1.3 Aynı Anda İşlemleyebilme

Parallel processing, bunu destekleyen donanım üzerinde, bir birine benzer işlemlerin aynı anda hesaplanabilme özelliğidir. Parallel processing yöntemi kendi içerisinde bir kümeleme yöntemi olabileceği gibi (genetik algoritmalar örneğini daha sonraki bölümlerden birinde vermiş bulunmaktayız), oluşan kümelerin aynı anda hesaplanabilirliği de kümeleme işlemlerinin faydalarından birisi olmaktadır. Çünkü girdi veri birbirinden bağımsız parçalara bölünebilmekte ve bunlar kendi içerisinde çözülebilmektedir. Her bir kümenin çözüm uzayı da tüm girdi verinin çözüm uzayından çok çok küçük olduğundan çözüm kolaylaşmış olmaktadır.

Aynı anda işlemleyebilme kendi içerisinde çok büyük bir konudur. Ayrıca son dönemlerde, donanım hız iyileştirme işlemleri sınırlarına dayandığından, hız artırımı tek donanım üzerinden değil de, birbirine benzer donanımları aynı anda kullanmaktan geçmektedir. Daha önce yazılan programlar, donanımdaki iyileştirmeler sayesinde, kendiliğinden bu hızlandırmalardan faydalanabilir ve yapılan işlemler yeni bir donanıma geçtiğinde daha hızlı yapılabilir. Bu programlar çoğunlukla tek thread kullanarak yazıldığından, yeni donanımlarda bulunan birden çok çekirdek iyileştirmelerinden kendiliğinden faydalanamazlar. Birden çok çekirdekten faydalanmak kendiliğinden olacak bir iş değildir ve özel bir çaba ile sağlanabilmektedir. Güzel bir şekilde aynı anda işlemleyebilme uygulamaları yazabilmek gerçekten de zor bir iştir. Bunu destekleyen donanım üzerinde uygulamayı geliştirmeyi öğrenmek uzun zaman almakta, ayrıca çoğu programlamacılar bunu gerçekleştirmede zorluk yaşamaktadırlar.

Bu kısımda aynı anda işlemleyebilme hakkında kısa bazı bilgiler verip, bu çalışmanın konularından biri olan büyük verinin de ilgilendirdiği bir kısım olduğundan açıklama ihtiyacı olmaktadır.

NOT: Tüm tez içerisinde, parallel processing ve distributed processing terimleri benzer amaçlarla kullanılmıştır ve ikisi arasında bizim bakışımız açısından bir farklılık olmadığı varsayılarak konular işlenmiştir. Gerçekte ise aynı anda işlemleyebilme ve dağıtık işlemleme birbirinden farklı konulardır ve çalışmada bu ayrıma girilmemiştir.

Aynı Anda İşlemleyebilme Hakkında Kısa Bir Özet

Aynı anda işlemleyebilme, birçok hesaplama işleminin aynı anda bir donanım üzerinde hesaplanma yöntemidir (Almasi, 1989). Amaç büyük bir işlemin birbirinden bağımsız parçalara bölünerek birbirinden bağımsız çalıştırılması ve aynı anda bitirilmeye uğraşılmasıdır.

Aynı anda işlemleme birden çok biçim üzerinde konuşulabilir;

- Bit temelli (bit level)
- Komut temelli (instruction level)
- Veri temelli (data level)
- Görev temelli (task level)

Bit Bazlı (bit level parelellizm)

Bilgisayar bilimleri tarihinde kendiliğinden yapılan hızlandırma işlemleri seksenli yılların sonu kadar çoğunlukla bit temelli aynı anda işlemleyebilme özelliklerindeki iyileştirmelerle sağlanmıştır (Culler, 1999). Bit temelli aynı anda işlemleyebilmeyi anlamak için işlemci içerisine girmek gerekmekte ve yapısını bilme ihtiyacı doğmaktadır. Herhangi bir işlemcinin aynı anda işleyebileceği veri miktarına word denmektedir. Bir kelime ise bitlerden oluşmaktadır. Bit olarak kelime uzunlukları da tarihsel olarak dört ile altmış dört arasında değişmektedir. Zamanımızda altmış dört bitlik makinalar yaygın olarak kullanılmaktadır. Kelime uzunluğu da işlemcinin yapabileceği ilk aynı anda işleme özelliğini belirtir. Örneğin iki on altı bitlik rakamı toplayacak bir işlemci, otuz iki bit ise iki arakamı aynı anda kelime dağarcığına alıp aynı anda işleyebilir. İşlemcinin aynı anda işleyebileceği veri miktarının da bir kelime olduğunu belirtmek durumundayız. Bir saniyede yapılan işlem miktarı ise bizim bu kelime üzerinde yapılan işlem miktarıdır ve bilgisayarın hızını belirtir.

İşlemciler daha önce tekbir çekirdekten oluşurlardı. Bu yüzden işlemci ve çekirdek sözcükleri çoğunlukla eşdeğer olarak kullanılırdı. Ancak birkaç on senedir, işlemci mimarisinde yaygınlaşan birden çok çekirdekli yapılar tek çekirdekli yapıların yerini almıştır.

Zamanımızda bilgisayarlar üstünde kullanılan çekirdekler yapı itibarı ile çoğunlukla serial işlemleme üzerinde kurulmuştur. Yani bir çekirdek aynı anda birden çok işlem yapamaz. Bunu yapabilen sistemler bulunmakla birlikte, her tarafta bulunabilen ve piyasada yaygınlık kazanan Von Neuman mimarisi sıralı hesaplama üstünde çalışır. Bizim bir bilgisayarlarda aynı anda yapılan birçok işlem aslında bir aynı anda çalışma benzetimidir ve bir yanılsamadır. İşlemci sadece her bir işleme bir zaman aralığı verir ve bu zaman aralığı her bir işleme belirli bir değnekleme (secheduling) sonucunda atanır ve aynı anda hesaplama gibi görünmektedir.

Birden çok çekirdek barındıran sistemlerde ise durum biraz faklıdır. Bu gibi sistemlerde aynı anda yapılabilen birim işlem sayısı çekirdek sayısı kadardır.

Komut Bazlı (instruction level parelellizm)

Komut temelli aynı anda çalıştırma yöntemleri bir işlemci çekirdeğinin aynı anda kaç adet komut çalıştırabileceği ile ilgilidir. Bir işlemci komutu örneğin bir rakamın işlemcinin bir kayıt (register) bölgesine yüklenmesi olabilir. Bir işlemci aynı anda iki adet yükleme (load) işlemine izin veriyorsa, iki ayrı yükleme işlemi aynı anda çalışıp aynı anda işleme özelliği sunabilir.

Başka bir yöntem ise, sanal işlemci yöntemidir (virtual processor). Bu sistemlerde bir çekirdek için birden çok işlem yolu (assemly line) bulunmakta ve işlemci aynı anda birden çok işlemi yapıyor görünebilmektedir. Bu işlem yolu adedi genellikle iki ile sınırlanmıştır. Bunu şu şekilde düşünebiliriz. Bir fabrikada iki adet işleme yolu bulunsun. Bir robot bu iki yol arasında her iki işlem yolunda çalışıyor gibi düşünebiliriz. Bu kısım daha çok işlemci mimarisi ile ilgilidir ve üzerinde çok durulmayacaktır.

Veri Bazlı (data level parallellizm)

Bu aynı anda işleme mantığında, aynı işlem farklı veriler üstünde koşar. Grafik işleme birimleri (Graphic Processing Unit/GPU) bu sistemlere güzel bir örnektir. Bu sisteme aynı zamanda tek komut çok veri (Single Instruction

Multiple Data / SIMD) denmektedir ve genellikle vektörler üzerinden çalışmaktadır. MATLAB bu sistem üzerinde çalışan güzel bir örnektir.

Bir karakter dizisinin içerisinde bulunan tüm karakterlerin büyük harfe çevrimi de bu sistemlerin çalışma mantığına güzel bir örnek olabilmektedir. İşlem basittir. Bir karakterin ASCII sayı değeri alınıp ona otuz iki eklemek kadar basittir. Ama bu sistemi destekleyen bir mimaride bu işlem (yapıya bağlı olarak ayrıca hesaplanan cümlenin uzunluğuna bağlı olarak) tekbir seferde sonuca vardırılabilir.

Günümüz masaüstü bilgisayarları grafik işlemcileri bu sistem üzerinde çalışır

Sistem olarak genel bilgisayar işlemcileri üzerinde benzetimi yapılabilmekle beraber, Von Neumann mimarisinden farklıdır. Vektörle hesaplamaya uygun olduğundan da son zamanlarda sayısal hesaplamalar için çok sayıda yazılım kütüphanesi bilgisayar üstünde veri temelli aynı anda işlem özelliğine sahip donanım varsa onu kullanmak üzere kendini özelleştirebilmektedir. Donanım bulunmadığında ise asıl işlemciye donuş yapılıp işlemin benzetim ile sonuca varılır.

Görev Bazlı (task level)

Veri temelli aynı anda işleme yönteminden farklı olarak, aynı veri üzerinde farklı işlemlerin koşması şeklinde açıklanabilir. Günümüz masaüstü bilgisayarları merkezi işlemcileri bu sistem üzerinden çalışır.

Görev Bazlı ve Veri Bazlı Melez Yapılar

Büyük veri sistemlerinde daha sonra anlatacağımız üzere Map/Reduce (M/R) mantığı geniş bir yer tutmaktadır. Haritalama veya azaltma işlemleri aynı anda koşabilmektedir. Bu sistem aslında görev temelli ve veri temelli melez bir yapıya benzemektedir. Ancak biraz farklı bir durum olduğunu da kabullenmek gerekir. Çünkü haritalama ve azaltma henüz (2016) bilinen kadarı ile özelleşmiş donanım üzerinde çalışmamaktadır. Her zaman matris benzeri yapılar elde edilemeyeceğinden veri temelli işlemler Von Neumann mimarisi üzerinde simule edilmektedir.

2.2 Kümeleme İşlemlerinin Dezavantajları

Kümeleme yöntemleri çok kullanılmakla beraber değişik açılardan dezavantajları da bulunmaktadır. Her bir dezavantaj farklı operasyonlarla iyileştirmeye çalışılabilir ancak yine de aşılamayabilirler. Bunları şu şekilde sıralamak mümkündür;

- En iyi çözümden uzaklaşabilme
- Uygun bir kümeleme yönteminin bulunamaması
- Çözüm sonrası işlemlerin uygulanması

2.2.1 En İyi Çözümden Uzaklaşabilme

Herhangi bir problem setinde, en iyi sonucu bulmak teoride her zaman mümkündür. Ancak bu, tüm set için olası bütün çözüm kümelerini ele aldıktan sonra en iyi çözüme göre sıralayıp ardından en iyisini bulmaktan geçmektedir. Bulunan her bir çözüm üzerinde, daha önce anlatıldığı gibi, hesaplamalar yapmak gerekir ve işlem sonunda da en iyi sonucu tutmak gerekmektedir. Tamamen kombinasyon işlemidir. Bu işleme ayrıca kaba kuvvet yöntemi (brute force) de denilir.

Kombinasyon işlemlerinin kötü tarafı ise veri kümesi arttıkça hesaplama zamanının artması ve bekleme sürelerinin mantık sınırlarının dışına çıkması olmaktadır. Bu nedenle, kaba kuvvet yöntemi çok tercih edilen bir yöntem olmamaktadır. Sade girdi veri kümesi yeterince küçük ise işe yarayabilen bir sistemdir. Aksi durumda daha başka çözüm yolları aranmalıdır ve kümeleme yöntemi bunlardan birisidir.

Kümeleme yöntemi ile çok büyük bir ihtimalle en iyi çözümden uzaklaşılmış olunur. Çünkü girdi veri kümesi daha hızlı bir işlem görebilmek için belirli kıstaslara göre parçalara ayrılır ve çalışma her bir kümede ayrı ayrı olarak çalıştırılır. Çalışan her bir parça ise, diğer küme elamanlarının olması durumunda nasıl bir çözüm olması gerektiğini düşünmez ve çoğunlukla da en iyi çözüm diğer küme elamanlarının hesaba katılması ile bulunabilir.

Hiç bir kısıtlayıcı neden yoksa (zaman aralıkları, araç nokta kısıtları vb.), kümeleme en iyi çözüme oldukça yakın sonuçlar üretebilir. Ancak kümelenmiş bir verinin her bir kümesinin diğer kümelere yakın olan sınır noktalarında her

zaman küme değiştirebilecek noktalar bulunabilir. Bir nokta bir kümeleme işleminin ardından bir kümeye atanmış olsa bile başka bir kümeleme işlemi ardından başka bir kümeye atılabilir. Nokta ve coğrafya temelli kümeleme işlemlerinde durum böyledir. Merkez olarak seçilen bir noktaya uzaklığa bağlı olarak yapılan kümeleme işlemlerinde sınır noktalar, merkezden uzaklıklarına göre ve uzaklığın değerine göre küme seçimi yapar. Ancak kısıtlamalar hesaplamaya girdiğinde en iyi çözüm çok farklılaşır ve kümeleme bu durumda en iyi sonuca ulaşma açısından daha da zorlanır.

Durum daha sonra anlatacağımız heuristic algoritmalarda da böyledir. Sezgisel algoritmalar yukarda bahsedilen kombinasyonlu işlemlerin zorluğundan başka yöntemlerle en iyi sonuca yakın ve kabul edilebilir bir sonuç bulmak için çalışırlar. Bu algoritmalar da en iyi sonuç garantisi vermezler ve uyguladıkları yöntemlerin en iyi sonuçtan uzaklaşılmasına neden olduğunu kabul ederler.

2.2.2 Uygun Bir Kümeleme Yönteminin Bulunamaması

Daha iyi bir sonuca götürebilecek iyi bir kümeleme algoritması oluşturmak her zaman zordur. Otomatik yapılan işlemler, bazen başka bir kümede olması gereken noktaları başka kümelere kayıtlamış olabilmektedir.

Nokta yönelimli kümeleme işlemlerini düşünürsek, iyi bir lagoritma coğrafi koşulları hesaba katmalıdır. İki nokta arası çok yakın olabilmekte ve aynı kümede olması gerektiği düşünülebilmektedir. Ancak iki nokta arası coğrafi koşullar buna izin vermeyebilmektedir. Kuş uçuşu uzaklık bu yüzden her zaman en iyi sonuçlara götürmeyebilir. Başka bir örnek, iki nokta arası çok yakın olsa bile, noktanın çalışma zamanlarının birbirlerine olan uyumsuzluğundan dolayı yine aynı kümeye konması sorun çıkarabilmektedir. Yine bir noktaya gidebilen araç türleri diğer bir noktada işlem görebilecek araç türlerinden farklı ise yine aynı nokta aynı kümeye ayarlanmamış olmalıdır.

Bu çalışmada yukarda bahsedilen kümeleme problemlerinden çalışma zaman aralıkları ile ilgili soruna değinilmiştir ve bunun için bir çözüm sunmuş bulunulmaktadır. Nokta temelli uzaklığa dayanan kümeleme üzerine ayrıca uygulanan zaman aralığı temelli kümeleme işlemleri ile bu soruna başka bir açıdan yaklaşılmıştır.

Bu çalışmada konumuzun ispatını yapabilmek için diğer bir kısım problemlerin var olmadığını kabul ediimektedir. Yani kısıt olarak sadece zaman aralıklarının varlığı olduğu varsayımı ile gidilmiştir. Örneğin araç çeşitleri kısıtlarının olmadığını düşünmüş bulunmaktayız. Bu varsayım, diğer kısıtların da benzer bir şekilde aşılamayacağını göstermemektedir. Ancak başka bir çalışmanın konusu olabilir.

2.2.3 Çözüm Sonrası İşlemlerin Uygulanması

Bir öneki bölümde kümelemenin neden en iyi sonuca varmada sorun oluşturabileceğinden bahsetmiştik. Dolayısı ile tüm rotalama işlemleri sonrası, daha iyi bir sonuç elde edebilmek için, görece rafine edilmiş sonuç üstünden iyileştirmelere gitme ihtiyacı doğmaktadır. Bu nispeten kolay bir işlemdir. Yapılan denemeler, kombinasyon işlemlerinde olduğu gibi nerede ise sonsuz bir deneme yanılma ile değil de, zaten şimdiden oluşturulan rotalar üstünden çalışacağından, daha hızlı bitebilecek bir işlem olmaktadır.

Değişim işlemleri tiplerini şu şekilde belirtmek mümkündür;

- Bir rotadan alınan noktanın başka bir rotaya eklenmesi
- İki ayrı rota bulunan iki noktanın karşılıklı olarak değiştirilmesi
- Her bir işlem tipinde uygulanabilecek işlemleri şu şekilde sıralayabiliriz;
- Nokta değişimi
- Yük değimi
- Araç değişimi
- Başlangıç nokta değişimi
- Başlangıç zaman değişimi
- Daha kötü bir sonuç kabul edilip daha sonraki sonuçlarda iyileşme denemeleri
- Rotanın tamamen bozdurulup yeniden sadece bozulan kısım için ayrıca rotalama yapmak
- Rotanın tamamen bozdurulup açıkta kalan yüklerin var olan diğer rotalara eklenme işlemleri

Bu çalışmada bu işlemleri yapmamakla beraber, yapıldığı takdirde çok daha iyi sonuçların bulunabileceği kanısında bulunmaktayız. Şimdi kısaca yapılan bu işlemler açıklanmaya çalışılacaktır.

2.2.3.1 İyileştirme Çeşitleri

Tüm iyileştirme işlemlerini iki kısma ayırmak mümkündür ve her birisi aşağıdakilerden birisi veya ikisi içerisine alınabilir.

Bir rotadan alınan noktanın başka bir rotaya eklenmesi

Bir rotada bulunan bir yükün veya bir noktanın tüm yükleri ile birlikte başka bir rotaya taşınması işlemidir. Bu yapılırken de ekleme yapılacak rotadaki kısıtları göz önüne almak gerekmektedir. Aynı noktaya götürülecek birden çok yük varsa, bir rotadan başka bir rotaya yük taşıma her zaman iyileştirmeye neden olmayabilir ancak taşıma yapılacak rota sayısı birden fazla ise iyileştirme ihtimali de artar.

İki ayrı rota bulunan iki noktanın karşılıklı olarak değiştirilmesi

Bir önceki yönteme benzer ama tek farkı karşılıklı alış veriş mantığı vardır. Böylece bir önceki yöntemlere kısıtların elvermediği iyileştirmelere imkân verilir. Çünkü kısıtlar karşılıklı olarak nokta veya yük çıkarımı ile gevşetilir ardından karşılıklı olarak denemeler yapılır. Denemeler sonrası iyileştirme olursa kabul edilir.

2.2.3.2 İyileştirme Çalışmaları

Kümeleme yöntemi, daha önce anlatıldığı üzere, büyük bir çoğunlukla en iyi sonucu veremeyeceğinden ve iyileştirme için kapı aralayacağından, rotalama sonrası çıkan plan üzerinde birçok işlem yapılabilmektedir. Bunun için yapılabilecek işlemleri liste halinde yukarda belirttik. Şimdi ise her birisinden kısaca bahsedilecektir.

Nokta değişimi

Oluşan bir planlama sonucu elimizde her birisi ayrı bir araca ayarlanmış şekilde rotalar bulunacaktır. Bu çalışmada, rotalama ile ilgili çoğu literatür çalışmalarında da benzer şekilde uygulandığı üzere, tek tipte ve sınırsız bir araç adedi olduğu var sayılmıştır. Bu nedenle araç seçimi bir kısıt olmamakta, ancak aracın kapasitesi bir kısıt olmaktadır. Nokta değişimi işlemi de, oluşan tüm

rotalarda zaman aralığı ve kapasite sınırının izin verdiği ölçüde, bir rotada bulunan bir yükün başka bir araca eklenmesi ile denemeler yapmak üzerine dayanır. Herhangi bir araçta bulunan bir yük eğer o araçtan çıkarılırsa büyük bir ihtimalle aracın gideceği mesafenin de kısalması anlamına gelir. Çünkü daha önce uğraması gereken bir noktaya uğramayacaktır. Alınan bu yükün başka bir araca eklenmesi de, eklenen aracın alacağı mesafeyi uzatacaktır. Ancak bu işlem (bir araçtan çıkarıp başka bir araca ekleme) toplam planlamanın mesafesini kısaltma ihtimaline sahiptir. Bu nedenle nokta değişimi oluşan planlamaya katkı sağlayabilir. Ancak denildiği gibi değişim diğer kısıtlamaları bozmamalıdır. Şöyle bir örnek verebiliriz;

Uğramamız gereken üç adet noktanın olduğunu farz edelim;

A, B, C

Noktalar arası uzaklık matrisinin aşağıdaki Tablo 1'deki gibi olduğunu farz edelim.

Çizelge 2.1: Örnek Mesafe Matrisi

Başlangıç	**Bitiş**	**Mesafe**
A	B	1
A	C	2
B	A	1
B	C	2
C	A	2
C	B	2

İki adet rotamızın olduğunu kabul edelim.

1. A-B-C=1+2=3
2. B-C=2

Bu durumda toplam plan mesafesi 5 birim olacaktır. Her bir noktaya tekbir yükün olduğunu varsayalım ve eğer ikinci rotada bulunan noktayı birinci rotaya eklemiş olsa idik, toplam mesafe yine birinci rotanın mesafesi olan üç olacak ve maliyetimiz düşecekti. Çünkü rota zaten B noktasında C noktasına gider ve ayrıca bir nokta ekleme ihtiyacı bulunmaz. Böylelikle ikinci rota tamamen kaybolur.

Yük Değişimi

Nokta değişiminde, değişim yapılan noktaya taşınan tüm yükler bir rotadan diğerine eklenir. Yük değişimi işleminde ise, eklenebilecek yüklerden hepsinin veya bir kısmının taşınması işlemi yapılır. Yük değişim işlemi tek defa uygulandığında çoğu zaman bir iyileştirmeye neden olmaz. Ancak bazı yükler bir başka rotaya ve kalan yükler başka bir rotaya eklenip işlem devam ettirilirse planda iyileştirme olma ihtimali bulunmaktadır.

Araç Değişimi

Bir rotaya araç ayarlama işlemi iyileştirme işlemlerinden biridir. İyi bir rotalama işleminde araç ayarlama işlemleri kendi açısından ayrı bir operasyondur. Araç ayarlama iyileştirmesi iki aşamada yapılabilir

- Rotalama sırasında
- Rotalama işlemi bittikten sonra

İki seçenekten birisini kabul etmek ise zaman ile ilgili bir seçim olmaktadır.

Rotalama Sırasında Araç Ayarlama İyileştirme İşlemleri

Rotalama sırasında rotaya bir araç ayarlanabilir ve bir kaynak olarak sadece o rotaya hizmet verecekmiş varsayımı yapılır. Rota için ayarlama yapıldıktan sonra da, araç kaynak havuzundan çıkarılır ve sonraki rotalar bu araç ile işlem yapamaz. Çünkü şimdiden kullanılmıştır.

Rotalama sırasında yapılan bu işlem çok yavaş bir işlemdir. Çünkü rotalama işlemleri genellikle kombinasyon işlemleridir ve çok büyük miktarda rota denemeleri içerir. Her bir deneme için ise bir araç ayarlayabilme uzun zaman alabilecek bir işlemdir. Tek bir rotaya ayarlama yapmak çok uzun sürmeyebilir, ancak denemeler yapılan tüm rotalar için toplam süre çok uzayacaktır.

Rotalama sırasında araç dolum optimizasyonu da yapılıyorsa bu süre çok daha uzayacaktır. Ancak yapılan işlem daha doğru bir işlem olacak, oluşacak planlama daha iyi bir sonuç verecektir.

Rotalama işlemi bittikten Sonra Araç Ayarlama İyileştirme İşlemleri

Yukarda bahsedilen gibi rotalama sırasında araç ayarlama işlemi pahalı bir işlemdir. Ancak planlama bittiğinde ise çok ucuz bir işleme dönüşür.

Rotalama sırasında yapılan işlemin maliyeti, yukarda kombinasyon işlemlerinin neden veri miktarı arttıkça çarpımsal olarak arttığını anlatılan kısımda denildiği gibi,

$$C(\mathrm{n}, \mathrm{r})$$

kadardır. Ancak rotalama işlemi sonrası maliyet çok düzer. Çünkü oluşan rota sayısı "n" ve araç sayısı "m" ise, n*m adet işlem yapılması gerekecektir. Ancak bu durumda da rotalar zaten oluşmuş ve iyileştirme seçenekleri azalmıştır. Başka bir sorun ise, araç dolum iyileştirme işlemlerindedir. Zaten oluşmuş bir rotada araç dolum ve yerleştirme işlemi problemlidir ve iyi sonuçlar vermez.

Başlangıç nokta değişimi

Literatürde kullanılan çoğu deneme veri kümeleri tek bir noktadan çıkar ve birden çok noktaya yük dağıtır. Bu tipteki veri kümeleri merkezi depo yapısı olarak adlandırılır. Ancak tüm rotalama işlemleri merkezi depo yapısına dayanmaz. Örneğin depolar arası rotalama işleminde herhangi bir depodan herhangi başka bir depoya yük taşıma gerekmektedir. Başka bir örnek ise, bir siparişin herhangi bir depodan çıktığı önemli olmayabilir. Bu tipte rotalama işleminde ise başlangıç noktanın değişimi nokta değişimi benzeri bir iyileştirmeye neden olabilmektedir.

Başlangıç zaman değişimi

Bir depodan çıkış zamanının maliyette iyileştirme yapabilmesi mümkündür. Bunun nedeni ise depodan çıkış saatine bağlı olarak, noktanın çalışma saatlerine uymadığı için gidilemeyen noktaların bulunması ihtimalidir. Bundan dolayı da depodan çıkış zamanı değişiminde rotada uzatma ve kısaltmalar olması ihtimalini taşır. Denemeler istenirse rastgele veya belirli bir sayıda yapılabilir. Başlangıç zamanından itibaren belirli aralıkla zaman ilerilere taşınarak işlem

yapılabilir ve rotanın bu başlangıç zamanına göre nasıl bir tavır alacağı belirlenebilir. Alınacak tavır ise kısıtlara bağlıdır. Örneğin belirli bir zamanda başlanılırsa bazı yükler alınamayabilir. Çünkü yüklerin ulaşma tarihleri bunu engelleyebilir. Bazı noktalara uğramada da sorunlar yaşanabilir. Çünkü gidilen noktaların çalışma zaman aralıkları çıkış zamanına ve o noktaya ulaşma zamanına uyumlu olmayabilir. Uyumlu olmayan noktalar ve yükler yerine yeni noktalar veya yükler eklenirse rotanın maliyet işlemleri değişebilir. Bulunan duruma göre de rota daha iyi ve kötü olarak kabul edilebilir. Ama bu varsayılan maliyet fonksiyonuna bağlıdır.

Daha Kötü Bir Sonuç Kabul Edilip Daha Sonraki Sonuçlarda İyileşme Denemeleri

İyileştirme çalışmaları sırasında greedy bir yöntem benimsenirse, her zaman daha iyi bir rota bulunduğunda kabul edilir. Ancak bu kabul rotalama veya rotalama sonrası iyileştirme sırasında daha kötü bir sonuca sebep olabilir (Yuce, 2016). Bu nedenle iyileştirme çalışması yaparken, o anda çıkan kötü bir sonucu kabul edip, daha sonar daha iyi bir sonuca varılacağı bir ihtimal içindedir. Simulated Annealing yöntemi böyle bir yöntemdir (Czech Z J, 2002) (Yuce, 2016). Oluşan planlama sonucuna böyle bir iyileştirme çalışması yapmak sonucun daha iyi yapılması ihtimalini taşır.

Rota Bozma İşlemleri

Rota bozma işlemleri rotalama sonrası uygulanan iyileştirme işlemlerindendir. Rotalama bittikten sonar beğenilmeyen rotalar daha iyi planlama yapılabileceği düşünülerek bozulabilir, ardından kendi aralarında tekrar rotalama işlemine sokulabilir veya var olan diğer rotalara eklenebilirler.

Rotanın Tamamen Bozdurulup Yeniden Sadece Bozulan Kısım İçin Ayrıca Rotalama Yapmak

Bozulan rotadaki yükler kendi arasında rotalama işlemine uğratılırsa, daha iyi bir planlama bulma her zaman olası değildir. Ancak çözüm kümesi çok küçük olacağı için hızlı bir işlem olmaktadır. Tüm girdi veri kümesi üzerinden değil de, sadece belirli bir algoritmaya göre seçilen belirli bazı rotalar üstünde uygulanır. Ucuz bir işlem olduğundan da uygulaması ve sonucu görme kolaydır. Bu nedenle de çok tercih edilen bir yöntemdir.

Rotanın Tamamen Bozdurulup Açıkta Kalan Yüklerin Var olan Diğer Rotalara Eklenme İşlemleri

Bir önceki yönteme benzerdir. Tek farklı yönü rotalama işlemi yapılmaması ve diğer rotalara ekleme denemeleri yapılır. Rotalama işleminden biraz farklıdır. Çünkü var olan rotalara ekleme yapılırken o rota üzerinde olan yüklere karışamaz. Bu yükler zaten var olduğundan kısıtlar izin verdiğince ve iyileştirme olabildiğince uygulanır.

2.3 Kümeleme Yöntemleri

Kümeleme yalnızca daha hızlı çözüm bulmak için yapılmaz. Beraber gitmesi gereken yüklerin kendi arasında kümelenmesi veya beraber hizmet verilmesi gereken noktaların kümelenmesi ayrı işlemlerdir.

Kümeleme yöntemlerini ikiye ayırabiliriz;

1. Rotalama dışı yapılan kümelemeler
2. Rotalama için yapılan kümelemeler

2.3.1 Rotalama Dışı Yapılan Kümelemeler

Rotalama öncesi, dağıtım öncesi bazı gereksinimler dolayısı ile çeşitli kümeleme benzeri işlemler yapılabilmektedir. Bunları aşağıda anlatacağımız şekilde anlatabiliriz.

2.3.1.1 Yer Rotalama (Location Routing)

Ulaşım firmaları için depoların kuruluş yerleri büyük önem arz etmektedir. Kurulum yeri ayrıca rotalama sonucunu etkileme bakımından da önemlidir. Örneğin, daha çok Doğu Anadolu'ya yapılacak gönderimler olacaksa, depo yerinin İstanbul olması çok anlam ifade etmeyecektir.

Yer rotalamada amaç her ne kadar çıkış yapılacak yerlerin dağıtım yapılan yerlere yakın olması düşünülse de, başka kısıtlar da için içine girebilmektedir. Örneğin vergi inşaat yapılacak yerin durumu, personel imkânları, coğrafi şartlar da önem verilen başka noktalardır.

Yer rotalama için bir örnek vermek gerekirse; Türkiye'nin her tarafına dağıtım yapan bir firma olduğunu düşünelim. Depolarını kurmak için yer

arayışındadırlar. Nokta olarak karşılarına iki ilimizin çıktığını düşünelim; Çorum veya Eskişehir. Çorum Türkiye'nin coğrafi olarak merkez noktalarından birisidir. Tarihte Hitit İmparatorluğunun neden başkentlerini burada kurmalarının bir sebebi de budur. Ancak nüfus olarak düşünüldüğünde büyük şehirlere olan uzaklığı meydandadır. Diğer taraftan Eskişehir ise coğrafi olarak merkez noktalara biraz daha uzak olabilir. Ama nüfus yoğunlukları ele alındığında tüm büyük merkezlere yakın bir yerdir. Bu nedenle, Eskişehir'in seçilme olasılığı daha yüksek olmalıdır.

Çalışmada bu konudan bahsedilmeyecektir.

2.3.1.2 Kısıtlama Yönelimli Kümelemeler

Rotalama öncesi var olan bazı kısıtlamalar, çeşitli kümeleme benzeri işlemlere neden olabilmektedir. Örneğin bazı araçların bazı noktalara hizmet vermesi istenmeyebilir. Bu nedenle belirli tipte kullanılacak araçlar kullanılacaksa, bazı noktaların kendi arasında kümelenmesi mantıklı olabilmektedir. Taşınan yüklerin çeşitleri de ayrı bir kümeleme işlemine sebep olabilmektedir. Soğuk veya sıcak taşınması gereken yükler kendi aralarında taşınacaksa kümeleme yöntemine başvurulabilir.

2.3.1.3 Problem Tipine Bağlı Olarak Parallel İşlenebilme Özellikli Kümelemeler

Rotalama içerisinde kümeleme işlemi yapılmamasına rağmen, problemin tipine bağlı olmakla beraber kümeleme benzeri bir yapı oluşturma imkânı bulunabilmektedir. Bunlar çoğunlukla kullanılan yönteme bağlı olmaktadır. Örneğin genetik programlamada, algoritma içerisinde bulunan kendiliğinden bir kümeleme işlemi bulunmaktadır. Rastgele bulunan bireyler oluşturulan popülasyona atılırken aslında bir kümeleme işlemine tabi tutulmaktadırlar. Her bir popülasyon aslında bir küme olarak kabul edilebilir. Benzer algoritmalar benzer yöntemler sunabilmektedir. Böylece kümeleme faydaları algoritmanın içerisinde varsayılan olarak kullanılabilmiştir.

Benzer şekilde bir algoritma kendi içerisinde parallel hesaplama yapıyorsa, kümeleme yapıyor olarak kabul etmemiz mümkün olabilmektedir. Ancak bu yöntem sadece algoritmaya bağlı olmakta ve yine başka kümeleme işlemleri algoritma üzerinde uygulanabilir durumdadır.

Çalışmada kullanılan yöntem kümelemeyi gerçek anlamı ile kullanmakta, yani işlem öncesi bir hesaplama ile girdi veri belirli kümelere bölünmekte ve bu kümeler parallel olarak kendi içerisinde en iyi yöntemi bulmaya çabalamaktadır.

2.3.2 Rotalama İçin Yapılan Kümelemeler

Rotalama için yapılan kümelemeler daha çok rotalama sırasında kullanılan kümelemelerdir.

2.3.2.1 Coğrafi Nokta Yönelimli Kümeleme

Coğrafi nokta yönelimli kümeleme, rotalama işleminin hızlı sonuç verebilmesi için yapılmaktadır. Bir rotalama probleminde bin adet nokta varsa, bu noktaları coğrafi olarak belirli bazı kümelere ayırmak ve bu kümeleri kendi aralarında rotalama yapılacak işlemi çok daha kısa zamanda bitirilmesini sağlayacaktır.

Bu kümeleme yöntemi ilk işlemde her zaman daha iyi bir sonuç vermeyebilir. İşlem sonunda toplu bir iyileştirme yapılan genel bir uygulamadır. Bunun nedeni, kümelemenin bir birine yakınlık metriğine göre benzer olan noktaları başka kümelere ayarlamasıdır.

2.3.2.2 Zaman Aralığı Yönelimli Nokta Kümeleme

Bu tezde sunduğumuz ve coğrafi nokta yönelimli kümelemeye ek olarak yapılan yöntemdir. Noktalar iki adet kümeleme işlemine uğratılır. Zaman aralığı kümelemesini birinci yöntem olarak veya ikinci yöntem olarak uygulayabiliriz. İlk aşama coğrafi yönelimli yapılıp ikinci kümeleme yöntemi olarak zaman yönelimli yöntem uygulanabilir.

Amaç şöyle açıklanabilir; Bir noktadan çıkış yapan bir araç belirli bir süre içerinde varabileceği noktalar kümesi vardır. Buna ek olarak da noktaların hizmet verebileceği zaman aralıkları bulunmaktadır. İlk kümeden alınacak yolun tutacağı zamana göre hizmet verilen zaman aralıklarına uymayan noktalar çıkarılır ve boşuna onları hesaba katması engellenir.

2.4 Kümeleme İşlemleri

Bu kısımda, daha önce yaptığımız açıklamalar üzerinde bir özet yapmak amacı taşınmıştır. Burada anlatılan tüm kısımlar, kümeleme ilgili olan diğer

bölümlerde de anlatılmıştır. Ancak, bütün bir resim çizebilmek adına, böyle bir bölüm ayırma ihtiyacı hissedilmiştir.

Bu kısımda anlatılanlar, ayrıca çalışma sırasında kullanılan kümeleme metoduna bir giriş olacaktır. Bir sonraki bölümde ise uygulamadaki şekli ile daha ayrıntılı bir biçim alacaktır.

Kümeleme işlemleri yapılırken aşağıdaki şekilde bir işlem sırası olduğu düşünülebilir.

1. İlk kümeleme
2. İkincil kümeleme işlemi
3. Kümelerdeki sınır bölge iyileştirmeleri
4. Her bir kümenin kendi içinde çözümü
5. Toplu iyileştirme denemeleri
6. Sonuç kabulü

2.4.1 İlk Kümeleme

İlk kümeleme işlemi girdi verinin hangi kıstaslara göre belirli veri kümelerine bölüneceğini belirtir. Tezde ispatlamak istenilen konu iki aşamalı kümeleme ile rotalama işleminin daha doğru şekilde yapılabileceğidir. Yani tek kümeleme işlemi çoğu zaman bazı kısıtlara takılacak ve daha iyi sonuçlar veremeyeceği üzerinde kurgulanmıştır. Yapılan iki ayrı kümeleme nokta ve zaman aralığı temelli kümelemedir. Daha farklı kümeleme yöntemleri de uygulanabilirdi ancak tez konusunu ispat edebilmek adına bunun yeterli olabileceği düşünülmüş. Ayrıca başka bir kümeleme işlemi kolaylıkla yapılan işlem üzerine eklemlenebilir. O zaman iki aşamalı kümeleme işleminden değil de, üç aşamalı kümeleme işlemi bahsedilmiş olacaktır.

İlk kümeleme işlemini yaparken dikkat edilmesi gereken bir nokta, bir sonraki kümeleme aşamasına yeterli büyüklükte bir veri gönderebilmektir. Örneğin bin adet nokta bulunan bir girdi veri kümesini yüz adet kümeye bölmek ikinci aşama kümeleme işlemi için güze sonuçlar vermeyecektir. Bunu deneme yanılma ile bulmak mümkündür. İkinci aşama kümeleme yeterince büyük bir veri kümesi ile yapılmalıdır. Bunu da deneme yanılma ile bulmak mümkündür.

2.4.2 İkincil Kümeleme İşlemi

Kümeleme işlemleri uygulanırken dikkat edilmesi gereken bir nokta bulunmaktadır. İki aşamalı kümeleme yöntemi aynı anda uygulanamaz. Bu şu demektir; iki aşamalı kümeleme arka arkaya yapılmalıdır. Önce nokta temelli kümeleme yapılıp ardından oluşan kümeler kendi içerisinde iç kümeler oluşturacak şekilde yine kümelenir. Verinin yeterince büyük olduğu düşünülüp bunun daha iyi sonuçlar vereceği varsayılmıştır.

Bin adet sipariş/yük barındıran bir girdi verimiz olduğunu düşünelim. Bu kümeyi öncelikle dört adet nokta yönelimli ilk kümeleme işlemine uğratalım. Bu işlem sonucunda, veriye bağlı olmak üzere, elimizde her birisi yaklaşık iki yüz elli adet nokta içeren kümeler bulunacaktır. Bu işlem bittikten sonra ise ikincil kümeleme işlemi başlayacaktır. İkincil kümeleme ise her bir kümeyi zaman aralığı yöntemi göre yine dört farklı kümeye bölecektir. Böylece toplamda 16 adet kümeye sahip bulunacağız. Coğrafi olarak birbirine yakın olmakla beraber, çalışma zamanı olarak da birbirlerine yakın olan noktalar bir kümede toplanmış olacaktır.

2.4.3 Kümelerdeki Sınır Bölge İyileştirmeleri

Sınır bölge iyileştirilmesi, her bir kümeleme işlemi ardından yapılabilecek bir işlemdir. İstenirse sadece nokta temelli kümeleme işlemi sonrasında veya zaman aralığı kümeleme işlemi sonrasında, ayrı ayrı veya sadece tek bir aşamaya uygulanabilir. Sınır bölge iyileştirmesi farklı kümelere atanan bazı noktaları daha uygun bir başka kümeye ekleme işlemidir. Örneğim K-Ortalama algoritması ile kümeleme yapılmışsa, bir nokta başka bir kümeye ait olması gerekirken diğer bir kümeye atanmış olabilir. K-Ortalama algoritması çözüm yaparken elinde sadece uzaklık bilgisi bulunmaktadır. Zaman olarak da kısıtlar varsa, bu gibi iyileştirmeler sonucu daha uygun hale getirebilir.

Aidiyet sorunu içinde istenirse el ile istenirse kendiliğinden çalışacak bazı yöntemler bulunabilir.

2.4.3.1 El İle Yapılan İyileştirmeler

El ile yapılabilecek iyileştirmeleri iki kısma ayırabiliriz.

1. Gözle kümelere bakılıp bazı noktaların başka bir kümeye atılması
2. Kümeleme işleminin tekrar çalıştırılarak uygun kümeler bulunana kadar denenmesi.

Gözle Kümelere Bakılıp Bazı Noktaların Başka Bir Kümeye Atılması

Tezde uygulanmayan bir yöntem olmakla birlikte bazı durumlarda sonuca daha hızlı götürebilen bir iyileştirmedir. Kümeleme sonucu göz ile takip edilebilecek durumda ise uygulanabilecek bir sistemdir. Bunun için masaüstü bir uygulama ve coğrafi sistemlere uygun bir uygulama gerekmekte ve taşıma ve bırakma özelliğine sahip olunmalıdır. Yanlış ayarlanmış noktaları sürükle bırak ile istenilen kümelere taşınıp bırakılması mümkündür. Diğer türlü excel üstünde bu işlemi yapmak zordur.

Kümeleme İşleminin Tekrar Çalıştırılarak Uygun Kümeler Bulunana Kadar Denenmesi

Çoğu kümeleme algoritması her çalışmada aynı sonucu vermez. Bunun nedeni çalışma sırasında bazı rasgelelik sistemlerinin bulunmasıdır. Örneğin K-Ortalama algoritması merkez noktayı her çalışma sırasında rastgele seçebilmektedir. Bu nedenle her defasında farklı yapıda kümeler elde edilecektir. Kümeleme işlemine yeterince zaman verildiğinde daha uygun sonuçlar verebilmektedirler ancak bu zamanın ne kadar olacağı veya kaç defa döngü yapılması gerektiği ayrı bir iyileştirme konusudur.

2.4.4 Her Bir Kümenin Kendi İçinde Çözümü

İlk üç işlem sonucunda elimizde bulunan kümeler için rotalama işlemi başlayabilir. Rotalama işlemi ile ilgili daha sonraki kısımlarda gerekli olan bilgileri vereceğimizden şimdilik bu kısım kısa tutuldu.

2.4.5 Toplu İyileştirme Denemeleri

Daha önceki bölümlerde anlatılan kümeleme ve rotalama ardından yapılan iyileştirme denemeleridir.

2.5 Uygulanan Yöntemler

Yukarda her bir rotalama işlemi ardından yapılabilecek iyileştirme işlemi sırasında mümkün olan ve yapılabilecek işlemler sıralanmıştır. Bu kısımda ise tezde nasıl bir yöntem izlenildiği anlatılacaktır.

2.5.1 İlk Kümeleme Ve İkinci Kümeleme

İlk kümeleme ve ikinci kümeleme işlemlerini yaparken iki türlü deneme yapıldı. İlk kümeleme önce nokta temelli denendi ve ardından zaman aralığı temelli olarak denendi. Nokta temelli olarak alındığından genel olarak daha iyi sonuçlar elde edildi ancak zaman aralığı temelli yapılan denemelerde de güzel bazı sonuçlar alındığı görüldü. Bu nedenle tezde gösterilen örnek en iyi sonuçlardan bazıları zaman aralığı kümelenin ilk önce uygulandığı sistemden, çoğu da nokta temelli ilk kümelemenin yapıldığı planlamalara aittir.

Kümeleme işlemi yaparken bir adet parametremiz bulunmaktadır ve kaç adet küme oluşturulmalı parametresidir. Kümeleme işlemi ardından yapılan rotalama hızlı olduğundan değişik değerde küme sayıları için de denemeler yapılmıştır. Ancak hangi sayıda küme için daha iyi sonuç bulunmuş olduğu kaydı tutulmamış ve başka bir çalışma için bırakılmıştır.

2.5.2 Her Bir Kümenin Kendi İçinde Çözümü

İlk kümelemenin hemen ardından bir rotalama denemesi yapıldı. Bu sonuç kaydedilerek, hemen ikinci kümeleme yapılarak rotalama işlemi her bir küme içerisinde bir daha yapıldı. Tüm kümelerde işlemler bittikten sonra tüm rotalar alınarak tek bir planlama olarak kabul edildi. Çıkan sonuç ilk kümeleme ardından yapılan rotalama işleminden daha iyi sonuç verdi ise kabul edildi. Yoksa diğer denemelere geçildi. Diğer denemeler olarak da, her bir kümeleme işleminin kaç adet küme çıkarması ile ilgili olan parametrenin değiştirilmesidir. İki aşamalı kümeleme yapıldığından da, her bir aşamada farklı bir parametre ile gidilebildi.

2.5.3 Kümelerdeki Sınır Bölge İyileştirmeleri Ve Toplu İyileştirme Denemeleri

Sınır bölge iyileştirilmeleri ve toplu iyileştirme işlemi tek bir işlem olarak ele alındı ve aynı anda aynı algoritma işlemlerine koyuldu. Yapılan iyileştirme denemeleri ise daha önce bahsedilen denemeler açıklamasına uygun olarak yapılmıştır. Ancak şöyle bir durum bulunmaktadır. Planlama işlemleri süresince yüzlerce rotalama deneyi

yapılmış olmasından, iyileştirme çalışması sadece en iyi rotalama sonuçları üstünde yapılmıştır. Çıkan her bir rotalama için yapılmamıştır.

2.5.4 Sonuç Kabulü

Bir sonucun kabul edilmesi için gereken koşul ise sadece mesafe bazında daha iyi bir sonuç üretebilmesi olmuştur. Başka planlama işlemleri sırasında başka koşullar gerekebilmektedir. Örneğin rotanın biraz uzun olmasına izin verilerek daha kısa zamanda dağıtımın bitmesi istenebilir. Tez için ise sadece mesafenin yeterli olacağı düşünülmüştür.

2.6 Kümeleme İşlemine Benzer Uygulanabilecek İyileştirme Yöntemleri

Kümeleme işleminin asıl amacı girdi verinin küçültülüp daha uygun bir çözüm kümesi oluşturmaktır. Kümeleme işleminin uygulanamayacağı zamanlar kümeleme işlemine benzer kullanılabilecek yöntemler de bulunmaktadır. Bu yöntemler kendi başlarına da kullanılabilir olmakla birlikte kümeleme ile birlikte de kullanılabilirler. Örneğin karınca kolonisi algoritması benzer bir yöntemdir. Kümeleme işlemi kendiliğinden oluşmakta ve en çok kullanılan rota parçaları kendi aralarında birleşip rota oluşturabilmektedirler. Heuristik Kabarcık Algoritması (Heuristic Buble Algorithm) da benzer bir yöntemler birbirlerine yakın yerlere gidebilecek yükleri alarak toplu olarak işlemleri bitirebilmektedir (Savran A. I., 2014). Daha pek çok örnek verebilmek mümkündür ancak şimdilik bu kadar bir açıklama ile bıraktık.

3 BÜYÜK VERİ NEDİR?

Büyük veri, geleneksel veri tabanı sistemlerinin yönetemediği veya yönetmekte zorluk yaşadığı çok büyük boyutlardaki verilerin bütünüdür (Lynch, 2008). Büyük veri, kelime anlamı olarak içerisine değişik formatlarda yapıları tutabilmektedir. Herhangi bir log dosyasından, çok büyük yazı yönelimli CSV (virgül ile ayrılmış) dosyaları da içermektedir (Holmes, 2012). Ayrıca zamanımızda oluşan sosyal medya çıktıları da büyük veri teknolojisine azımsanmayacak katkılar yapmıştır. Milyar sayıda kullanıcısı olan siteler, çıktılarını yönetebilmek için büyük veri yapılarına büyük yatırımlar yapmışlardır (Boyd, D. & Crawford, K., 2012).

Büyük veri sistemlerinin önemli bir özellikleri yatay genişleme yöntemine uygun olmalarıdır. Yapılan bir işin büyümesi nedeni ile kullanılan makinenin özellik olarak büyütülmesi yerine (dikey büyüme), aynı makineden bir adet daha kullanılmasına yatay büyüme denilir. Kullanılan makineler her tarafta bulunan genel makineler olunca da fiyat bakımından da uygun çözümler ortaya çıkabilmektedir.

Ayni veri miktarına sahip iki sistem olduğunu düşünelim. İlk sistem klasik yollarla, yani geleneksel veri tabanı sistemleri ile tasarlanmış olsun. Diğer sistem ise yatay büyüme ile oluşturulan bir "büyük veri" kümesi ile çalışıyor olsun. Veri miktarı büyüdükçe, ilk sistemin dikey olarak geliştirilmesi gerekmektedir. Kullanılan ilk makine yeterli gelmediğinde daha büyüğünü tercih etmek gerekecektir. İkinci sistemde ise, ilk makineden katlarla daha pahalı olabilecek ikinci bir makine kullanma yerine, aynı fiyata aynı makineden bir adet daha eklenerek işin yapılması sağlanacaktır.

Geleneksel veri tabanları, büyük veri kümelerinden esinlenerek bazı kümeleme yöntemleri sağlamaktadır. Ancak bu çözümler yine de büyük veriden alınabilecek verime yetişemeyecektir ve genellikle doğal olmamaktadırlar. Parçalama (sharding) buna güzel bir örnektir.

Çok sayıda büyük veri çözüm sağlayıcısı bulunmaktadır. Sunulan çözümler açık kaynaklı olabileceği gibi, kapalı yazılım çözümleri de bulunmaktadır. Bu dokümanda, en çok kullanılan açık kaynak büyük veri çözümlerinden biri olan Hadoop altyapısı kullanılmıştır. Ayrıca Hadoop üstünde çalışan çözümler olan Spark, Prediction.IO gibi yapay zekâ veya istatiksel sonuç üreten yazılımlar da kullanılmıştır.

3.1 Büyük Veri Tanımı

Büyük veri tanımı doksanlı yılların başından beri kullanılmasına rağmen ilk olarak John Mashey (Mashey, 1997) tarafından büyük kitlelere tanıtıldığı düşünülmektedir.

Büyük veri ilk kez Dug Laney tarafından üç adet madde ile tanımlanmıştır (Laney, 2001);

1. Veri miktarı
2. Veri hızı
3. Veri çeşitliliği

3.1.1 Veri Miktarı

İşletmeler herhangi bir kullanıcı işlemi sonucunda, işlem ile ilgili daha fazla veri tutmak isteğindedirler. Ancak veri miktarı arttıkça da, bunun masrafı çoğalmaktadır. Büyük veri sistemleri bunun için bir çözüm sunmalıdır. Veri miktarı arttıkça masrafın dikey olarak mümkün oldukça az artması sağlanmalıdır. En azından kabul edilebilir bir düzeyde olmalıdır. Bununla masraf düştükçe daha fazla veri saklanmak istenecektir. Bu da kullanıcılar hakkında daha fazla bilgi sahibi olunması ve verilen hizmetlerin iyileştirilmesi anlamına gelecektir.

Geleneksel yollarla toplanan verilerde genel olarak toplanan veri miktarı artıkça verinin değeri düşmekte ve tutulması için de verilen önem düşmektedir. Nedeni ise bu verinin yönetiminde yaşanabilecek sorunlardır. Büyük veri sistemleri buna bir çare olabilmedir. Veri işlemesi kabul edilebilir zamanlar aralıklarında bitirebilmelidir.

3.1.2 Veri Hızı

Herhangi bir anda gelen veri miktarı bir önceki duruma göre katlanarak çoğalmaktadır. Bu miktarın geleneksel yöntemlerle yönetilebilmesi imkânsız olmaktadır. Problem sadece bant genişliği de değildir. Gelen verinin uygun zamanda işlenebilmesi anlamına gelmektedir. Bu yapılamadığı sürece toplanan verinin bir anlamı da olmamaktadır. Çünkü anlık veriler belli bir süre sonunda anlamsız olabilmektedir. Gerçek zamanlı büyük veri işlenebilmesi ayrıca rekabet aracıdır. Daha hızlı ve güncel veriler sunabilme, kullanıcılar açısından, tercih sebebi olabilmektedir.

3.1.3 Veri Çeşitliliği

Teknoloji ilerledikçe veri çeşitliliği de artmaktadır. Web sayfalarından alınabilecek XML (Extensible Markup Language/Geliştirilebilir İşaretleme Dili) veya JSON (Javascript Object Notation / Javascript Nesne Yazımı) yapılarından tutun da, küçücük duyarca verilerine kadar bir yelpazede toplanabilecek yapılar bulunmaktadır. Bunlar aynı zamanda yapısal veya yapısal olmayan gelişi güzel bir şekilde toplanabilmektedir.

3.2 Bir Büyük Veri Sisteminin Parçaları

Her hangi bir büyük veri sisteminin bazı parçaları bulunmaktadır. Bunları şu şekilde sıralayabiliriz;

1. Donanım/Yazılım Düzeyi ve Sistem Kullanımı
2. Dağıtık bir dosya sistemi
3. Bahsedilen dosya sistemi üzerinde dağıtık hesaplama yapabilecek bir sistem
4. Konu edilen dağıtık hesaplama sistemi üzerinde çalışabilecek uygulamalar.

3.2.1 Donanım/Yazılım Düzeyi ve Sistem Kullanımı

Büyük veri sistemleri en alt düzeyde disk kullanımı sırasında daha çok okuma kapasitesi (transfer rate) üstünde çalışır ve arama (seek time) kapasitesi üzerinde çok çalışmamaya dikkat ederler. Bunun sebebi arama kapasitesinin her geçen yıla nazaran daha çok artması ve arama zamanının fiziksel bir işlem

olduğundan çok da gelişmemesidir. Geleneksel veri tabanı yönetim sistemleri, B-Tree benzeri yapılarla bu arama zamanına bağlıdırlar. Disk üzerinde okuma kafasının bir noktaya ulaştırılması fiziksel olduğu için de yavaştır. Örneğin bir disk bir saniye içerisinde on adet kafa yeri değişikliği yapabilirken, okuyabildiği veri miktarı bazı disk yapılarında yüz megabayt olabilmektedir ve bu miktar her geçen yıl daha da artmaktadır.

Ayrıcı sistem düzeyinde Hadoop altyapısını diğer sistemlere kullandırabilecek bir altyapının da bulunması gerekmektedir. Örneğin çeşitli yazılımlar HDFS sistemi kullanacaklarsa bunu bir büyük veri kütüphanesi yardımı ile yaparlar. Yapılacak her bir işlem için yardımcı bir yordam bulunacaktır.

3.2.2 Dağıtık Dosya Sistemi

Büyük veri sistemlerinin kalbinde dağıtık bir dosyalama sistemi (Distributed File System / DFS) bulunmaktadır. Bu sistemleri herhangi bir Windows makinasında bulunan dosyala sistemi ile karşılaştırabiliriz. Dosyaların saklandığı bilgisayarlar birden çok makine üstünde dağılmış bulunmaktadır. Örneğin bir terabaytlık bir dosya 10 adet makine üstünde belli bir sayıda çoğaltılmış olarak bulunabilmektedir. Ancak, dağıtık dosya sistemi kullanıcıya tek bir yapı olarak görünmektedir. Görüntü olarak herhangi bir dosya sisteminden farkı olmamakla birlikte yapı olarak çok farklıdır (Shvachko, 2010).

Dağıtık dosya sistemlerinin en önemli özelliklerinden birisi de hataya karşı korunma sistemleridir. Örneğin herhangi bir dosyayı barındıran bazı makineler kapanmışsa ve diğerleri açıksa sistem yine de çalışmaya devam edebilmektedir. Büyük veri sistemleri piyasada çokça bulunan nispeten ucuz sunucu yapıları kullandıklarından, hataya karşı duyarlı olabilmeleri çok önemli olmaktadır (Shvachko, 2010).

3.2.3 Dağıtık Hesaplama Sistemi

Büyük veri sistemlerinin herhangi bir parçası genel olarak yeni bir yapı değildir ve yıllar öncesinden gelen bir birikim ile başka sistemler üstünde kullanılmıştır. Dağıtık hesaplama da böyle bir yapıdır. Ancak parçaların birleşimi ile zamanımızdaki veri miktarına uygun teknolojiler geliştirilebilmiştir.

Dağıtık hesaplama sistemi olarak çok farklı sistemler kullanılmakla birlikte, büyük veri sistemlerinde genellikle M/R yapısı kullanılmaktadır. Tanım olarak yeni bir tanım olmamakla birlikte (fonksiyonel programlama dillerinde genel bir tanımdır), dağıtık hesaplamada kullanım alanı bulmuştur.

Google 2003 yılında kendi kümelerinde çok büyük verileri nasıl yönetebildikleri hakkında bir makale yayınlamış (Ghemawat, 2003) ve daha sonra oluşturulacak yazılımlar için bir ilham kaynağı olmuştur. Bunun devamı niteliğinde olan ayrı bir makale ile de (Dean, 2004) yılında büyük veri dağıtık hesaplama sistemlerinin önünü açmıştır.

Yöntem olarak veri parçalara bölünmekte, her bir parça uygun olan bir makinada haritalamaya maruz bırakılmaktadır. Haritala gerekli olan verinin bir işleme sokulmasıdır. Böylece kullanıma hazır hale getirmedir. Ardından gerektiğinde bu veri azaltma işlemi ile veri özetlenebilmektedir. Azaltma işlemi de herhangi bir veri parçası için herhangi bir makinada yapılabilmektedir. Sonuç olarak da, oluşan veri üstünde başka bir haritala/azalt işlemi çalıştırılabilmekte veya dağıtık dosya sistemine daha sonra kullanabilmek amacı ile kaydedilebilmektedir. Görüldüğü gibi verinin tamamının herhangi bir makinanın belleğine sığacak büyüklükte olması gerekmemekte ancak belli bir parçasının sığabilmesi yeterlidir. Böylece yapılan işlemler binlerce makineye dağıtılabilmekte ve kısa sürede donanımın etkili bir şekilde kullanımı ile (taşıma kapasitesi kullanılarak) bitirilebilmektedir.

3.2.4 Büyük Veri Sistemi Üzerinde Çalışabilecek Uygulamalar

Bu bölümde şu ana kadar anlatılan kısımlar bir büyük veri sisteminin altyapılarını oluşturma ile ilgilidir. Ancak büyük veri sistemlerinin asıl değerini, üstünde çalıştırılabilecek uygulamalar belirlemektedir. Bu uygulamalar, veri tabanı benzeri yazılımlar olabileceği gibi (NoSQL vb.), yapay zekâ hesaplama yazılımları (Prediction.IO) da olabilmektedir. Hepsi de büyük veri sisteminin verdiği imkânları kullanarak hesaplamalarını gerçekleştirmektedir.

3.3 Bir Büyük Veri Sistemi Olarak Hadoop

Bir büyük veri sistemi olarak Hadoop, 2003 yılında yayınlanan Google makalesinden (Ghemawat, 2003) esinlenerek büyük geliştirmeler göstermiştir. Hadoop geliştiricileri, onu açık kaynak olup genel ve her tarafta bulunan makinalarla kurulan makina kümeleri üstünde çalışan dağıtık hesaplama ve dağıtık depolama sistemi olarak tanımlamıştır (Apache Yazılım Vakfı, 2016). Her yer yerde bulunabilen genel sunucular kullanılması, hem ucuzluk anlamına gelmekte hem de hataların/makine bozulmalarının çokça meydana gelmesi demek olduğundan, Hadoop geliştirilirken göz önüne alınan önemli bir etken hataların nasıl yönetileceği olmuştur.

Hadoop çoğunlukla Java programlama dili ile yazılmakla birlikte bazı kısımlarının C dili veya uygulama ek kısımları olarak da kabuk programlama dili ile yazıldığı bilinmektedir (White, 2015).

3.3.1 Hadoop'un Genel Bir Tarihi

Hadoop üstünde çalışmaya, Apache Lucene yaratıcısı Doug Cutting başlamıştır. İlk basamakları Apache Nutch içerisinde şekillenmiştir. Apache Nutch, açık kaynak bir web arama motoru yazılımıdır ve Apache Lucene içerisinde konumlanmıştır (White, 2015). İsim olarak Hadoop seçilmesinin sebebini de kızının oyuncak filine verdiği ad olarak açıklar. Kendisi ile birlikte yol alan arkadaşı Marc Caferalla ile birlikte, o zamanlar, bir milyar sayfa barındıran bir indeksleme yazılımının yarım milyon dolar donanım ile ve aylık otuz bin dolar işletim masrafı ile yapılabileceğini düşünmekteydiler. (MIKE CAFARELLA, 2004)

İlk olarak 2002 yılında üstünde çalışmaya başladıkları Nutch ile birlikte, sistem ilk sürümü ile ortaya çıkmıştır. Fakat görülen bu sistemin o andaki hali ile tüm web için bir arama motoru altyapısı olması imkânsızdı. 2003 yılındaki Google makalesi ile birlikte, Nutch Dağıtık Dosya Sistemi (Nutch Distributed File System / NDFS) üzerinde çalışmaya başladılar. Bu onları tüm webi indeksleme yolunda sonsuz bir depolama donanımı yönetimi yükünden kurtarmışa benzer.

2004 yılında Google yeni bir makale daha yayınlamış, M/R mantığının kendi sistemlerinde kullanımını anlatmıştır. 2005 yılında ise Nutch ekibi çalışan bir H/A sistemini oluşturmuşlardı.

2006 yılına gelindiğinde Nutch projesinde bulunan NDFS ve H/A kısımları ayrı bir proje altına alınarak Hadoop ismini aldı.

Doug Cutting Yahoo! için çalışmaya başlayınca, Yahoo!, Lucene projesinin ana sponsorlarından biri oldu ve 2008 yılında arama indekslerini on bir adet makine bulunduran bir küme ile Hadoop altyapısını kullanarak çalıştırır.

2009 yılında, Hadoop bir terabayt bilgiyi altmış iki saniyede sıralayabildiğini belirtmiştir (O'Malley, 2008). Aynı işlemin Google tarafından altmış sekiz saniyede yapıldığı belirtilmiştir (Sorting 1PB with MapReduce, 2008).

3.3.2 Hadoop Sistem Bileşenleri

Hadoop sisteminin genel bileşenleri şu şekilde belirtilebilir;

1. Hadoop Common: Tüm Hadoop sistemini destekleyen bir altyapı kütüphanesi
2. Hadoop Distributed File System (HDFS):Yüksek hacim sağlayan bir dağıtık dosya sistemi.
3. Hadoop YARN: İş değnekleme ve küme kaynak yönetimi kütüphanesi
4. Hadoop M/R: YARN üstünde çalışan büyük veri kümelerini işleme olanağı sağlayan sistem.
5. Hadoop Çevre Yazılımları: Uygulamaları: Hadoop üstünde koşabilen ve ona yardımcı olabilecek işlevler sağlayan uygulamalar ve yazılımlardır. Genel olarak ve çoğunlukla kabul görmüş olanlar aşağıdaki listede belirtilmiş olan uygulamalardır;
    - Ambari™: Hadoop sistemi kurulum ve yönetim sistemidir
    - Avro™: Hadoop sisteminde iletişim altyapısı ve dosya yapısı ile ilgili bir kütüphanedir
    - Cassandra™: Hadoop üstünde koşabilen bir NoSQL sistemidir
    - Chukwa™
    - HBase™: Hadoop üstünde koşabilen bir NoSQL sistemidir
    - Hive™: Hadoop üstünde koşabilen bir NoSQL sistemidir. Kendi SQL dili bulunmaktadır.

- Mahout™: Hadoop üstünde makine öğrenimi desteği sağlayan yazılımdır.
- Pig™: Hadoop'a dinamik programlama ortamı sağlayan bir programlama dilidir.
- Spark™: Hadoop üstünde de çalışabilen, Hive altyapısını kullanabilen M/R eşleniğidir.
- Tez™: Hadoop üstünde de çalışabilen, Hive altyapısını kullanabilen M/R eşleniğidir.
- ZooKeeper™: Dağıtık sistem koordinatörüdür.

Yukardaki ana bileşenlerin her birisi "Bir Büyük Veri Sisteminin Parçaları" bölümünde anlatılan kısımların her birisine karşılık gelir. Bir eşleştirme yapmak gerekirse aşağıdaki gibi belirtilebilir;

1. Donanım Düzeyi ve Kullanımı ⇔ Hadoop Common
2. Dağıtık bir dosya sistemi ⇔ Hadoop Distributed File System (HDFS)
3. Bahsedilen dosya sistemi üzerinde dağıtık hesaplama yapabilecek bir sistem ⇔ Hadoop YARN + Hadoop M/R
4. Konu edilen dağıtık hesaplama sistemi üzerinde çalışabilecek uygulamalar ⇔ Hadoop Çevre Yazılımları: Uygulamaları

Bu kısımlardan en çok üzerinde duracağımız kısımlar dağıtık dosya sistemi ve dağıtık hesaplama sistemleri olan Hadoop HDFS ve Hadoop M/R + Hadoop YARN kısımlarıdır.

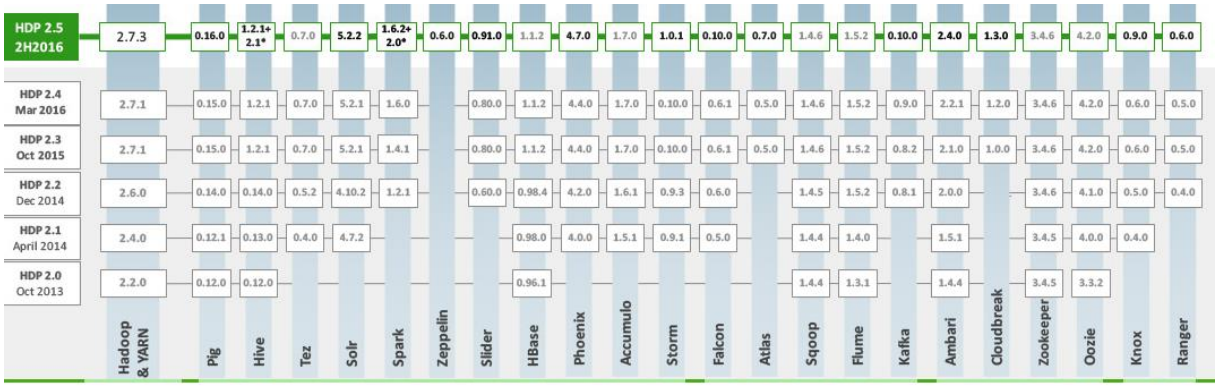
3.3.3 Ambari™

Hadoop sistemi birbirinden bağımsız olarak çalışan yüzlerce sistemden oluşmaktadır. Her bir sistem başka geliştirme ekipleri tarafından oluşturulmakta ve her birisi belli bir olgunluk kıvamına oluştuğunda Hadoop yazılım dağıtımına kabul edilmektedir. Birbirinden bağımsız bu kadar parçanın kurulum işlemleri de el ile yapılacaksa çok sancılı olabilmektedir. Özellikle bu uygulamalar birbirleri ile bağımsız olarak çalışıp aralarındaki iletişimi ise http üzerinden küçük hizmetlerle yaparlar. Bunun için de hizmet port ayarlamaları çok büyük bir önem arz etmektedir. Genel olarak kabul edilen portlar bulunmakla birlikte, farklı firmalar Hadoop dağıtımı hazırlarken bu portları değiştirebilirler. Başka bir zorluk ise, Hadoop kurulumu çevrebirimleri ile farklı makineler üzerine

yapılabilmektedir. Bazı yazılımlar bir makineye kurulurken başka uygulamalar da makinenin kaynak uygunluğuna göre başka makinelere ayarlanabilmektedir. Böylece Hadoop kurulumu el ile yapılırsa büyük bir işlem olmaktadır. Özellikle her bir uygulamanın tek başına alınıp derlenerek kurulumuna geçilmesi ile yapılacaksa sürüm uyumsuzluğu denen bir soruna neden olmaktadır. Bazı uygulamalar diğer bazı uygulamaların belli bazı sürümleri ile uyumlu olmasından da bu gibi sorunlar kurulumun yine büyük problemlere gebe olmasına neden olmaktadır. Ambari yazılımı bu problemlerin önüne geçmek için tasarlanmış bir sistemdir. Ambari ile Hadoop kurulumu belli bir dereceye kadar sorunsuz birden çok makinaya Hadoop ve çevre birimleri kurulumunu gerçekleştirir. Tezin hazırlandığı tarihte Windows temelli işletim sistemlerine uyumlu olmamakla birlikte Linux üstünde kurulum çok kolay bir şekilde yapılmaktadır.

Ambari sistemi olarak kullanılan sürüm Hortonworks firması tarafından dağıtılan sürümüdür. Bundaki tercih sebebimiz de Hortonworks'ün Windows'ta çalışabilen (Ambari ve birkaç uygulama hariç) tek sürüm olmasıdır. Tek tek her bir uygulama Windows için ayrıca derlenip ayarlanabilmekle beraber toplu kurulum imkânı verebilen tek sürüm olmasıdır.

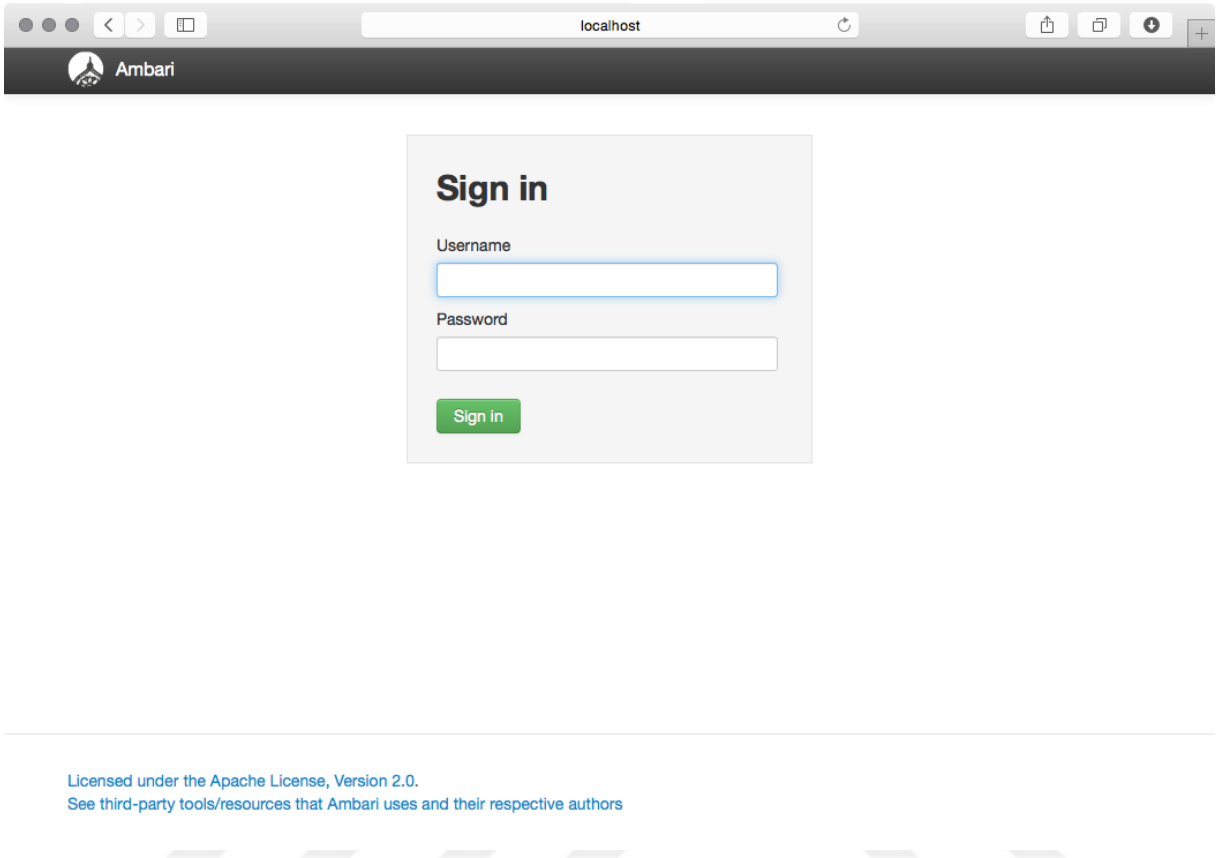
Şekil 3.1'de Ambari'nin kurulumunu yapabileceği Hadoop ve çevrebirimlerinin ve sürümlerinin toplu bir kesiti bulunmaktadır.



Şekil 3.1: Her bir Ambari sürümünde bulunan Hadoop çevrebirim sürümleri

Ambari uygulamasının bir başka özelliği ise kurulum ardından yönetim imkânı vermesidir. Ayrıca kurulu olan makinelerin sağlık durumu, hizmetlerinin açık olup olmadığı da yönetilebilecek işlemler arasındadır.

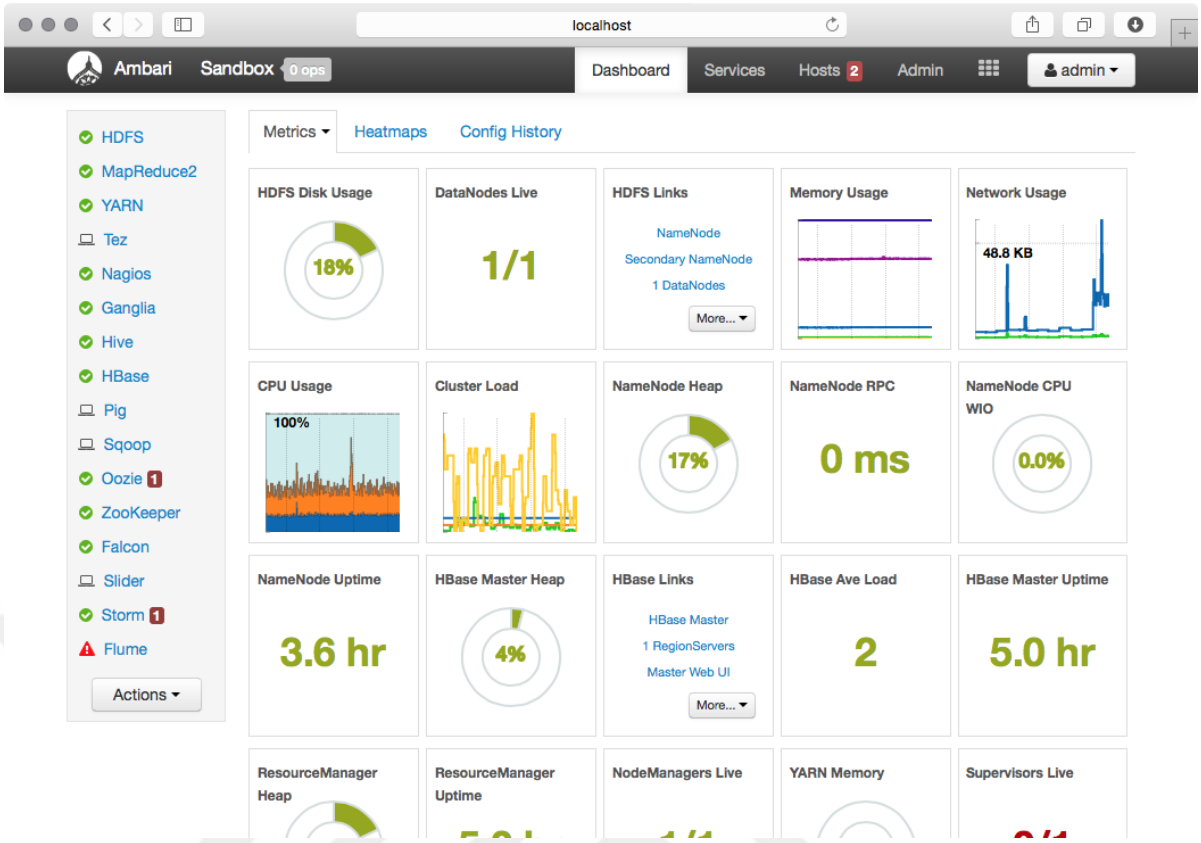
Şekil 3.2'de Ambari uygulamasının giriş ekranı ve genel bir ilk görünümü verilmiştir.



Şekil 3.2: Ambari giriş ekranı

Tez de kullanılan Hadoop kurlumu da Ambari ile yapılmıştır. Üç adet makine bulunmaktadır. Ambari uygulamasının kendiliğinden önerdiği Hadoop yazılımlarının makinalara dağıtımını kabul ederek kurulum yapıldı. Her makine birbirine benzer ve Ubuntu 14.4 işletim sistemi sürümünü içermektedir.

Sekil 3.3 ile Ambari uygulamasının genel bir görünümü olan ilk sayfası görülmektedir. Solda Hadoop ekosisteminde bulunan ve kuruluma dâhil edilen yazılım parçaları bulunmaktadır. Her birisine tıklanarak ilgili parçanın yönetim bölümüne gidilebilir. Şekilde görülen sayfa ana sayfadır ve her bir parça ile ilgili özet bilgi vermeye çalışır.



Şekil 3.3: Ambari ana ekranı

Ambarisiz yapılan Hadoop kurulumları üç veya dört gün sürerken, Ambari ile yapılan Hadoop kurulumları bir gün içerisinde bitmiştir. Bu bile Ambari kullanımının ne kadar önemli bir yöntem olduğunu göstermektedir.

Kullanılan ve Hortonworks'ün dağıtımını yaptığı Ambari sürümünde Spark kullanımı biraz zor olmaktadır. Özellikle uygulama içerisinden programlama dilleri kullanarak Spark kullanımı için ek ayarlamalar yapmak gerekti. Örneğin ODBC Spark ODBC sunucusunu kurmamız için gereken yazılımı Spark'ın da yazılımcıları olan DataBricks şirketinin web sitelerinden edinip sunucu olarak Ambari kurulumu üstünde kurulan Spark sürümüne bağlamak durumunda kalındı.

3.3.4 Avro™

Avro Hadoop'un da ilk geliştiricilerinden olan Doug Cutting tarafından geliştirilmeye başlanmış ve kullanılabilir bir yapıya getirilmiş bir iletişim ve taşıma sistemidir. Doug Cutting Hadoop sisteminde ilk olarak Java altyapısında bulunan Java uzaktan metot çağrımı sistemini kullanmaya çalışmış ama bunun Hadoop sistemine uygun olmadığına karar vermiştir. Ardından Google'ın ilk

geliştiricisi olduğu protobuf ile devam etmek istemiş ama onda da bulunan ve daha çok C programlama diline uygun olan yapısı nedeni ile onu da çok kullanılamaz bulmuştur. Daha sonra Hadoop sistemine uygun olabilecek başka bir sistem geliştirmek için, Cloudera firmasında çalıştığı sırada Avro geliştirmesine başlamıştır. Şu anda Hadoop sisteminde aktif olarak kullanılmaktadır. Avro hem bir dosya yapısı şeklinde hem de bir iletişim formatı olarak da kullanılabilmektedir. Yapı olarak tek bir dosyadan oluşan bir veri tabanı sistemi dosyasına benzemekle beraber Hadoop için iyileştirilmiş bir formatı bulunmaktadır. Kendi içerisinde tuttuğu veri yapısı ile ilgili bilgiler (şema) tutmakta ve veriyi en küçük şekli ile tutabilmek için veri sıkıştırma algoritmaları kullanmaktadır.

3.3.5 Cassandra™

Çalışmada kullanılmamakla birlikte Hadoop ekosisteminde kullanılmakta olan çevre uygulamalarından birisidir. Daha çok veri tabanı eşleniği olarak düşünülerek oluşturulmuş bir uygulamadır. Önemli özelliklerinden birisi merkezi bir yapısı olmamasıdır. Veri kendiliğinden çeşitli şekillerde çoğaltılır. Herhangi bir makinenin hata vermesi üzerine başka bir makineden hizmet verebilecek bir şekilde düzenlenmiştir. Bir NoSQL veri tabanı olmasına rağmen yine de verilerin sorgulanabilmesi amacı ile SQL benzeri bir dil kullanmaya başlamıştır.

3.3.6 Chukwa™

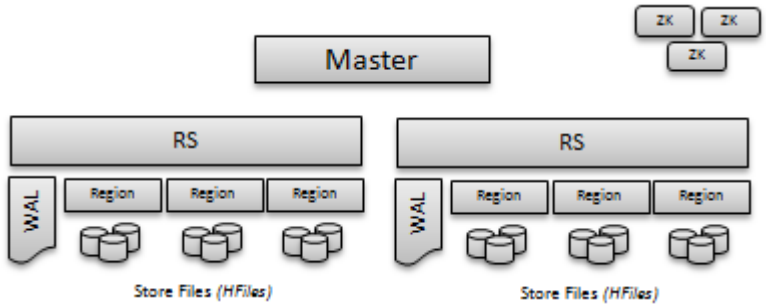
Chukwa Hadoop sistemi içerisine veri yükleme amacı ile oluşturulmuş bir yazılımdır. Çeşitli toplayıcılar üzerinden çalışır ve dışarıdaki veriyi HDFS üzerinde Hadoop araçlarının kullanabileceği bir yapıda bırakır.

3.3.7 HBase™

Cassandra uygulaması ile aynı piyasaya hizmet etmekle beraber birkaç farkları vardır. Bunlardan birincisi Hbase merkezi bir sistemdir. Merkez makinada bir problem olduğunda tüm sistem çalışamaz duruma gelecektir. İkinci olarak da Hbase hızlı sorgulamalar yapılabilecek bir sistem olarak kullanılabilir. Yapı olarak buna uygundur. Cassandra da ise daha çok uzun sürebilecek arka arkaya çalışacak işlemler koşturmak mümkündür.

Hbase uygulamasının başka bir özelliği de yazma işleminin çok iyileştirilmesi olmuştur. Yazma işlemi garantilenmiş ve çok hızlı bir şekilde tek bir makine ile yüzlerce GB'lik daha bir gün içerisinde yönetilebilmektedir.

Hbase çalışma mantığına kısa olarak değinmek gerekirse; Hbase, Şekil 3.4'de belirtildiği gibi, bir adet ana sunucu (master) ve çok sayıda bölge sunucusu (region) oluşmaktadır.



Şekil 3.4: Hbase yapısı

Bölge sunucuları da kendi içerisinde birden bölge barındırabilir. Her bir bölge ise birden çok kayıt dosyası (store file) içerebilmektedir. Her bir kayıt dosyası yazma işlemi yapılırken önce hızlıca olarak bellek ile haritalanmış (memory mapped) dosyalar üzerinde kaydedilir ve müşteri uygulamaya yazma işleminin bittiğini belirtir. Ancak, HBase arkada bu dosyaları kendi formatı olan kayıt formatına çevirir.

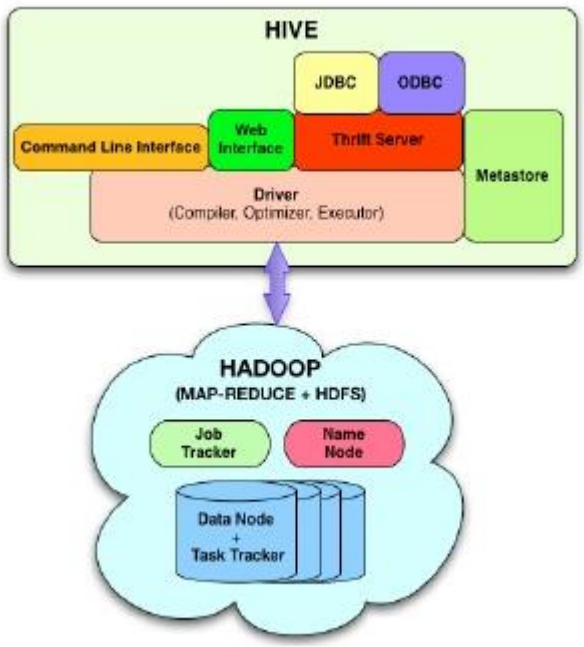
HBase'in hızlı olmasının bir sebebi de bazı vakit alabilecek işlemlerin daha sonra makinelerin uygun olduğu zamanlara bırakılmasıdır. Örneğin her yazma işlemi sonucunda kayıt dosyaları oluşturulur ama bu kayıt dosyaları aynı veri için birden çok kopya barındırabilmektedir. Uygun olan bu zamanlarda bu kopya veriler tekbir kopyada olacak şekilde birleştirilir.

3.3.8 Hive™

Hadoop sisteminin en çok bilinen yazılımlarından birisidir. Hadoop üzerinde sorgu yapabilmek amacı ile ilk sorgulama dilini oluşturan yazılımdır. İlk başlarda Facebook firmasında geliştirilmiş olmakla beraber şu anda açık kaynak haline getirilmiş bir sistemdir. Yapı bilgisi tutmak (metastore) için ilişkisel bir veri tabanı ihtiyacı bulunmaktadır. Bu veri tabanı Derby gibi küçük bir uygulama olabileceği gibi, Microsoft SQL Server uygulaması da olabilir.

Bu yapı bilgisi Hadoop sisteminde bir sorgulama dili veya benzerleri kullanılacaksa çok önemlidir. Örneğin Spark kendi sorgu dilini sunarken bu yapı bilgisini Hive üstünden kullanmıştır. Ayrıca Hive kendi içerisinde Hadoop'un varsayılan olarak sunduğu M/R sistemini kullanır. Ancak Spark bazı ayarlamalar ile Hive kullanırken altyapı olarak kendini kullandırabilir. Bunların hepsinin mümkün olmasını sağlayan bu yapı bilgisi tablolarıdır.

Hive'ı Hadoop üzerinde çalışan bir veri tabanı sistemi olarak da kabul edebiliriz. Ancak Hive buna ek olarak HDFS üzerinde olan herhangi bir dosyayı tablo olarak görebilir. Bu dosya üzerinde istenilen sorgulamalar yapılabilir. Şekil 3.5, Hive'ın Hadoop ekosistemindekini yerini ve bağlantısını göstermektedir.



Şekil 3.5: Hive Yapısı

Başka bir özelliği ise daha ilişkisel veri tabanlarından veri okuyabilmesidir. Bunu da dışsal tablo (external table) mantığı ile yapar. Örneğin Hive, Oracle veri tabanında bulunan bir tabloyu kendine bağlayabilir ve sanki bu tablo HDFS üzerinde imiş gibi onun üzerinde sorgulama yapabilir. Ancak bu tablonun bir dışsal tablo olarak yapı bilgisi veri tabanına kayıt edilmesi gerekir.

Hive barındırdığı tablolara çok farklı yöntemlerle ulaşım sağlar. Bu yollar ile farklı tipteki kullanıcılar bağlanıp istedikleri sorguları koşturabilirler. Yukardaki resimde görüldüğü gibi Java/Scala programlama dilleri veya

benzerleri (JDBC) ile iletişim kurabilirler. ODBC yöntemi de çok kullanılan ve evrensel olarak kabul görmüş bir yöntemdir. Diğer bütün programlama dilleri veya uygulamalar bu yöntem ile Hive'a bağlanabilirler.

Hive'ın çalışma mantığı da ilgi çeken bir konudur. Altyapı olarak Spark kullanılmıyorsa, Hive gelen her bir sorguyu M/R işlemlerine çevirir. Her bir işlem parallel olarak Hadoop üzerinde bulunan birçok makinaya bitirilmesi için dağıtılır ve işlem tüm makinelerde bitirilince birleştirilip müşteri uygulamaya sunulur.

Bu çalışma hazırlandığı sırada, Hive altyapı makinesi olarak üç farklı makine yazılımız kullanabilmiştir;

- M/R: Hadoop ortamında varsayılan hesaplama makinesi yazılımıdır.
- Spark: DataBricks firmasının geliştirdiği ve M/R'den daha hızlı olan yazılımdır.
- Tez: Hortonworks tarafından geliştirilen ve M/R'den daha hızlı olan yazılımdır.

Her birisi için kısa açıklamalar, yerleri geldikçe belirtilecektir.

3.3.9 Mahout™

Mahout ilk çıkışı itibari ile Hadoop sistemi üzerinde çalışabilecek makine öğrenimi ve yapay zekâ yazılımları olarak düşünülmüştü. Birden çok makine üzerinde aynı veriyi kullanarak tasarlanmış algoritmalar bütünüdür. İçerisinde K-ortalama gibi kümeleme algoritmaları bulunduğu gibi, önerme sistemleri (colloborative filtering/recommendation) de bulunduran bir sistemdir. En sonra Spark bünyesinde başlatılan Spark mllib içerisine alınmış veya bazı kısımları sadece mllib parçaları kullanarak yazılmış hale getirilmiştir.

Şua anda aktif olarak geliştirilmesi durdurulmuştur veya mllib üzerine birleştirilmesi devam etmektedir. Konumuzla ilgili olması nedeni ile Mahout ile ilgili kısımlar Spark Mllib kısmını anlatacağımız bölüme bırakılmıştır.

3.3.10 Pig™

Hadoop sisteminde kullanılmak üzere tasarlanmış bir derlenmeyen bir dil olarak düşünülmüştür. Daha önce Hadoop üstünde kullanılabilir dillerin azlığı nedeni

ile çok büyük bir önem verilirken şu anda Spark ve Python gibi dillerin de kullanılmaya başlanması ile popülerliği biraz da olsa azalmış görünse de henüz geliştirilmesi devam etmektedir.

3.3.11 Spark

Birkaç yıl önceye kadar M/R performansı kimse tarafından sorgulanmamakta ve tüm kullanıcılar tarafından kabul görmekte idi. 2009 ve sonrasında ise M/R üstünde bazı eleştiriler gelmeye başlamıştır. Bunun üzerine Hadoop geliştiricileri M/R sürüm 2 altyapısını kurmuşlar ve bunu Hadoop içerisinde sunmaya başlamışlardı. Ancak geçen zaman bunun da yeterli olmayacağını ve dağıtık hesapla kısmı üzerinde daha fazla bir performans isteğini açığa çıkarmıştır. Microsoft, Cloudera ve Hortonworks firmaları her birisi kendi özel dağıtık hesaplama sistemlerini geliştirmeye başlamışlar veya geliştirilen ve buna yardımcı olabilecek yazılımları desteklemişlerdi. Microsoft DryAdd üzerine çalışmalara başlamış (Li, 2013), bundan etkilenen geliştiriciler ise Spark üzerinde bağımsız olarak çalışmaya başlamıştır. Hortonworks ise Tez yazılımını yapmış ve bunun reklamını yapmıştır. Cloudera ise Spark üzerinde destek vermeye ve gelişimine yardım etmeye başlamıştır.

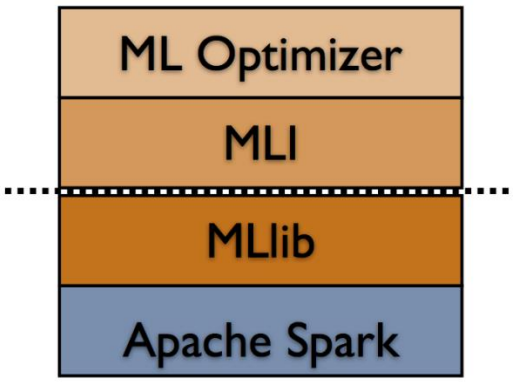
Spark son yıllarda büyük bir ivme kazanmış ve Hadoop sisteminin bazılarınca vazgeçilmez bir altyapısı olmuştur. Kolay kullanımı programlama mantığı ile büyük bir kitle tarafından kabul görmüştür.

Spark sistem olarak da ilginç bir yapıya sahiptir. Spark kütüphanelerini Hadoop sistemine bağlamaksızın da kullanma imkânı bulunmaktadır. Her ne kadar yapı olarak Hadoop ekosisteminde bir yazılım olsa da bu özelliği ile diğer birçok Hadoop altyapı veya üst yapısından farklıdır. Çoğu Hadoop yazılımı ise Hadoop sistemi olmadan çalışamaz ve işleyemezler. Örneğin Tez de bir Hadoop ekosistem ürünüdür. Kendisini bir M/R makinası olarak tanıtır ama Hadoop olmadan üzerinde işlem yapılamaz.

Spark Mllib

Daha önce Mahout kısmında açıklandığı gibi, Mllib konumuzla ilgili olduğundan üzerinde biraz durulacaktır. Spark yazılımı geliştirildikçe yapay zekâ ve makine öğrenimi kısımları ayrı bir proje olarak devam edilmiş Spark'ın bir alt projesi olarak Mllib adını almıştır. Tez'imizde örneğin mllib K-ortalama

algoritmasını çoğu zaman birincil kümeleme algoritması olarak kullandık. Mllib yazılımını diğer yapay zekâ yazılımlarından ayıran özelliği ise dağıtık hesaplama işlemleri için iyileştirilmiş olmasıdır. Şekil 3.6, Mllib ile Spark arasındaki bağlantıyı göstermektedir.



Şekil 3.6: Mllib'in Spark sisteminde yeri

Mllib üç ana bölümden oluşur

1. Öneri makinaları (Recommendation Systems / Colloborative Filtering)
2. Kümeleme (Clustering)
3. Sınıflandırma (Classification)

Öneri makinaları

İnsanların beğenileri çok değişim gösterebilir. Öneri işlemleri bu gibi değişiklikleri göz önüne alıp insanların bile fark edemeyeceği önerileri onlara sunabilir. Bu tamamen bir beğeni tahmininden ibarettir. Örneğin, web sitelerinde bolca gördüğümüz "beğenebileceğiniz ürünler" kısımları mllib gibi yazılımlar kullanılarak oluşturulur. Klasik öneri yazılımlarının takıldığı kısımlar ise girdi veri kümesinin çok büyük olduğu durumlardır. Mllib bu konularda devreye girer ve uygun sonuçlar üretmeye çalışır. Verinin dağıtık olduğu düşünülerek de kendini ona göre iyileştirmiş olur.

Bir öneri makinası üç kısımdan oluşur;

1. Model
2. Kullanıcı Benzerliği (User Similarity)
3. Kullanıcı komşuluğu (User Neighbourhood)

Model öneri işleminin üstünde koşacağı veri kümesini barındırır. İçerisinde kullanıcı ve tercih bilgileri bulunur. Kullanıcı benzerliği ise kullanıcıların

birbirine ne kadar benzediğini anlamaya yarar. Bunu yaparken de değişik algoritmalar ve kısıtlar kullanabilir. Kullanıcı komşuluğu ise bir kullanıcı grubunun ne kadar diğer bir kullanıcıya yakın olduğunu anlamaya çalışır.

Öneri makinaları sonuç olarak da bir kullanıcın herhangi bir ürüne ilgisini skor belirterek çıktı yapar

Örnek vermek gerekirse, daha önce yapılmış bir uygulama çıktısını ele alabiliriz. Girdi veri kümemiz Çizelge 3.1'de olduğu gibi düşünülsün;

Çizelge 3.1: Öneri makinesi örnek girdi modeli

Kullanıcı	**Ürün**	**Oran**
1	101	5
1	102	3
1	103	2.5
2	101	2
2	102	2.5
2	103	5
2	104	2
3	101	2.5
3	104	4
3	105	4.5
3	107	5
4	101	5
4	103	3
4	104	4.5
4	106	4
5	101	4
5	102	3
5	103	2
5	104	4
5	105	3.5
5	106	4

İlk kolon, kullanıcı numaralarını belirtmektedir. İkinci kolon ürün numarası ve üçüncü kolon ise beğeni miktarıdır. Her bir kullanıcı bir ürüne verilmek üzere bir den beşe kadar bir skalada beğenisini belirtmiştir.

Kullanılan kod parçası aşağıdaki gibidir;

```
class RecommenderIntro {
 public static void main(String[] args) throws Exception {
        DataModel model = new FileDataModel (new File("intro.csv"));
        UserSimilarity similarity =new PearsonCorrelationSimilarity (model);
        UserNeighborhood neighborhood =new NearestNUserNeighborhood (2, similarity, model);
        Recommender recommender = new GenericUserBasedRecommender (model, neighborhood, similarity);
        List<RecommendedItem> recommendations =recommender.recommend(1, 1);
        for (RecommendedItem recommendation : recommendations) {
                System.out.println(recommendation);
        }
 }
}
```

Bu kodda dosyada barınan bir model kullanılarak, kullanıcı benzerliği için de Pearson Correlation yöntemi ile en yakın on komşulukta çözüm bulmaya çalışmıştır.

İşlem sonucunda çıktı olarak şöyle bir veri sunulmuştur;

- Önerilen ürün; [ürün:104, değer:4.257081]
- 104 kitabı 1 kullanıcısına önerilmiştir.

Öneri makinesi bu şekilde davrandı çünkü kullanıcı 1'in, 104 kitabi için, tercihinin 4.3 civarı olacağını düşündü. Ayrıca bu tüm verilebilecek öneriler içerisinde en yüksek beğeni değeri idi.

Genel olarak algoritma şu şekilde çalışır;

- **Kullanıcı Temelli Öneriler Yapılacaksa**

1. u kullanıcısının önceden bir beğeni belirtmemiş olduğu her bir ürün i için
    a. Daha önce ürün i için beğeni belirtmiş her bir v kullanıcısı için
        i. u ve v arasındaki benzerlik s sayısını bul
        ii. v kullanıcısın i için beğenisini s ile katsayıla ve bir ortalama olacak şekilde hesapla
2. katsayılı ortalama üzerinden sıralanmış şekilde en yüksek n adet ürünü döndür.

- **Ürün Temelli Öneri yapılacaksa**

1. u kullanıcısının önceden bir beğeni belirtmemiş olduğu her bir ürün i için
    a. u kullanıcısının önceden bir beğeni belirtmiş olduğu her bir ürün j için
        i. u ve v arasındaki benzerlik s sayısını bul
        ii. u kullanıcısın i için beğenisini s ile katsayılayarak ve bir ortalama olacak şekilde hesapla
2. Katsayılı ortalama üzerinden sıralanmış şekilde en yüksek n adet ürünü döndür.

Kümeleme ve Sınıflandırma

Kümeleme işlemlerinin daha önce bahsedilen kümeleme işlemlerinden tek farkı, mllib içerisindeki algoritmalar dağıtık hesaplamaya uygun olan veya getirilen algoritmalar olmasındadır. Daha önce konu üzerinde durulduğundan şimdilik devam edilmedi. Ayrıca sınıflandırma da tez konusu ile ilgili olmadığından açıklanmadı.

3.3.12 Tez™

İlk adımları M/R altyapısının yavaşlığından yakınan ve Hortonworks'de çalışan mühendisler tarafından atılmıştır. Belirli bazı durumlarda Spark'dan daha hızlı işlem gördüğü iddia edilmektedir. M/R altyapısına Spark'dan daha yakın olmakla birlikte, yazılımcıları tarafından veri miktarının çok çok büyük olduğu

durumlarda daha iyi olduğu söylenmiştir. Her ne kadar Spark Hadoop'dan bağımsız olarak çalışıyor olsa da, Tez tek başına değil de, Hive sistemine entegre olarak arka planda çalışmaktadır.

Tez sırasında "Apache Tez" kullanılmaya çalışılmış ama kararlılık problemleri nedeni ile vakit kaybetmemek amacı ile bırakılıp Spark üzerinde devam edilmiştir. Ayrıca zaten çoğu algoritma Spark üstünden deneneceğinden üzerinde çok durulmamıştır.

3.3.13 ZooKeeper™

Hadoop sisteminde her bir uygulama bir hayvan ismi ile anıldığından, orkestrasyonu yapacak uygulamaya da hayvanat bahçesi koruyucusu adını vermek uygun olmuş olsa gerektir.

Asıl amacı dağıtık koordinasyon sunucusu olmaktır. Hadoop sisteminde birçok uygulama ZooKeeper ile dağıtık şekilde duran makinelerle veya makinalar üzerinde koşan uygulama parçaları ile konuşmaktadır.

Her bir uygulama başlatılınca kendini ZooKeeper ile register eder ve başka parçaların kendini bulmasını sağlar. Diğer uygulamalar da Zookeeper'a belirli aralıklarla sorgular gönderip dönen sonuca göre haberleşmesi gerekebilen dağıtık uygulamaları ve nasıl haberleşmesi gerektiği bilgisini alır ve işlemine devam eder. Örneğin Hbase, bölge sunucuları için Zookeeper'a sorgu atar. ZooKeeper da cevap olarak o ana kadar kendisine kayıt yaptıran bölge sunucularını ve iletişim bilgilerini (varsa) döndürür ve HBase ana sunucu da kendi içinde kayıt yapıp gerekli gördüğünde bu bölge sunucuları ile iletişime geçer.

3.4 Hadoop Dağıtımları

Hadoop dağıtım olarak bir kaç şirket tarafından sunulmaktadır. Açık kaynak olmakla birlikte, bir dağıtım kullanma ihtiyacı bulunmaktadır. Bunun sebebi de, birbirinden bağımsız olarak geliştirilen ve çalışan yüzlerce alt uygulamanın bir arada sunulması ve ayarlanması ihtiyacıdır. Böylece her bir yazılım tek başına kurulup ardından Hadoop ekosistemine her birisi ayrı ayrı birleştirilmiş olmayacaktır.

Hadoop merkez kaynağı Apache Yazılım Vakfıdır. Tüm geliştirmeler ve değişiklikler bu vakıf tarafından düzenlenir. Ancak dağıtım yapan firmalar, seçenekli olmak kaydı ile kendi özel değişikliklerini kendi dağıtımlarında sunmak şansına sahiptirler. Ama bu kabul gören bir davranış olmamaktadır ve sizi tek bir dağıtıma belli bir ölçüde bağımlı bırakacağından uzak durulması gereken bir seçenektir. Kullanılan altyapılar böyle değişiklikleri içermemektedir.

Her bir Hadoop dağıtımı ayrıca Hadoop işletme desteği vermekteler. Bununla birlikte kendi dağıtımlarını kendi kurulumlarında kullanmaktadırlar. Bu dağıtım hazırlanırken de kendi ihtiyaçlarına bağlı olarak da hangi Hadoop alt yapı uygulamasının uyumlu olduğu hangi sürümünü dağıtıma ekleyeceklerine kendileri karar verirler. Bunun sebebi de, Apache Yazılım Vakfı her aday sürüm çıkardığında bu sürümün dengeli bir yapıya kavuşmasını beklemektir. O yüzden her bir dağıtımın içerisinde her bir uygulamanın en son sürümü bulunmayabilir.

Tez yazımı sırasında piyasada en çok kullanılan üç adet dağıtım bulunmaktaydı ve bunlar aşağıdaki gibidir;

- Cloudera
- MapR
- Hortonworks

3.4.1 Cloudera

Cloudera kendi Hadoop dağıtımına “Hadoop’un Cloudera Dağıtımı” (İngilizcesi Cloudera’s Distribution of Hadoop - CDH) demektedir. Bu dağıtımda yukarda bahsedilen tüm açık kaynak araçları bulunmakla birlikte, kendi yazdıkları, ücretli ve kapalı kaynak olan bazı yazılımlar da eklemektedirler. Bu yazılımlardan olan Cloudera Yöneticisi (Cloudera Manager) uygulaması Ambari eşleniğidir ve kurulum, yönetim işlemlerini kolaylaştırır. Ayrıca Cloudera en çok kullanılan Hadoop dağıtımıdır ve en büyük kar marjına sahip şirket olarak durmaktadır. Müşterileri arasında Cisco ve Samsung da bulunmaktadır (Cloudera Customers, 2016).

3.4.2 MapR

MapR yine standart olan araçları bulunduran bir dağıtım yapmakla beraber, bunların yerine geçebilecek ve yüksek performanslı olduğunu belirttiği araçları da sunmaktadır. Örneğin M/R parçalarının yeniden tasarlanmış bir şeklini kendi dağıtımında kullanmak üzere eklemiştir ve sadece HDFS kullanımı ile değil başka dosya sistemlerini de Hadoop ortamına eklediğini iddia etmektedir. Amerikan Express ve Ericsson müşterileri arasındadır. En çok kar yaptığını düşünülen ikinci Hadoop dağıtımcısı firmadır.

3.4.3 Hortonworks

Yüzde yüz açık kaynak olan tek dağıtımdır. Yaptığı tüm geliştirmeleri yine Apache Yazılım Vakfına gönderir ve bu vakfın Hadoop projelerinde en büyük yardımcılarından birisidir. Şirket olarak da destek, eğitim ve kurulum hizmetleri vermektedir.

3.4.4 Diğer Dağıtımlar

Başka birçok irili ufaklı dağıtımcı şirket veya vakıf bulunmaktadır. Bunlardan en dikkat çekeninin Amazon Web Hizmetleri olduğunu söylemek yanlış olmaz. Kendi bilişim merkezlerinde kendi dağıtımları olan bir Hadoop kümesi kurmak mümkündür. Bundan başka Apache Yazılım Vakfının kendi sitesinde belirttiği onlarca dağıtımcıyı görmek mümkündür (Dağıtımlar ve Ticari Destek , 2016).

3.5 Bir Hadoop Dağıtımı Olarak Hortonworks Data Platform

Kısaca HVP olarak isimlendireceğimiz Hortonworks veri platformu, çalışma sırasında altyapı olarak kullanacağımız Hadoop dağıtımının ismidir. Tez yazılırken son sürümü 2.5 idi ve kurduğumuz sürüm de bu oldu. HVP'nin bir Hadoop dağıtımı olarak seçiminin yapılmasının sebebi ise Ambari gibi bir kurulum ara yüzü sunmaları ve ayrıca bunun için ücret istenilmemesi idi. Cloudera ayrıca bir ara yüz sunmakta ancak bunun için önemli bir miktarda ücret talep etmekte idi. Ayrıca Windows işletim sistemine uyumlu olması da bu tercihe neden olmuştur. Her ne kadar Windows işletim sistemine kurulum yapmadı isek de, her ihtimale karşı böyle bir seçim yapılmıştır. Bununla birlikte asıl kullanmakta olduğumuz Spark yazılımına tam destek vermemesi önemli bir

dezavantaj olarak görünmekle birlikte bunu daha önce anlatılan DataBricks ODBC sunucusu müşterisi kurularak aşmış bulunmaktayız. Ayrıca piyasada en çok kullanıcıya sahip olmaması da ayrı bir dezavantaj olmakta ama tez süresince bir problem yaşanmadığından değişim ihtiyacı hissedilmedi.

4 PROBLEM

Rotalama iyileştirme işlemlerinde, iki aşamalı (veya n aşamalı) kümeleme işlemi çok kullanılan bir yöntem değildir. Kümeleme işleminin yapabileceği bazı çıkarımları kısıt temelli olarak işlemlendirme genel yöntemdir. Bu aşağıdaki Çizelge 4.1'de belirtildiği gibi uygulanmaya çalışılır;

Çizelge 4.1: Rota başına kısıt kontrol algoritması

Adım	**İşlem**
1	İyileştirme yaparken, kontrol edilecek bir rotaya bakılırken, bu rotanın bir kısıta uyup uymadığı kontrol edilir.
2	Kısıta uyumlu ise kabul edilir ve maliyeti hesaplanır.
3	Çıkan maliyet hesabına göre daha iyi bir sonuç ise elde tutulur değilse atılır.
4	Ardından diğer rota adayları için bakılır.

Burada bulunan problem, girdi veri miktarı arttıkça bu işlemin en çok zaman alan işlem olasıdır. Yapılan işlemler de çoğu zaman kombinasyon işlemi olduğundan bu kısmın iyileştirilmesi gerekmektedir. İyileştirme için de iki adet yöntem sunulabilir;

1. Algoritma teknik olarak incelenir, daha hızlı çalışması için, yazılım yapılan programlama dilinde, iyileştirmeler yapılır.
2. Algoritmaya giriş yapılıp hesap edilen nokta sayısı düşürülmeye çalışılır.

Birinci yöntem ne kadar iyileştirilirse iyileştirilsin, sonuç genel iyileştirme uygulaması üzerinde büyüyebilme açısından veya asimptotik analiz açısından, çok büyük katkılar yapmayacaktır. Bununla birlikte kısıt sayısı birden çok

olabilir ve her bir kısıt çok farklı ve yavaş işlemler yapıyor olabilir. Tezde her bir kısıtın birim bir iş yaptığı düşünülmüştür. Gerçek hayatta ise bu böyle değildir. Kısıtlar çok büyük miktarda hesaplama yapıyor olabilirler. Ancak kodun iyileştirilmesi her zaman kabul edilen ve uygulanan bir yöntemdir.

Çalışmada konusu olan nokta azaltma işlemi, çözüm kümesi küçültme olduğundan her zaman daha hızlı bir biçimde sonuca götürecektir. Örneğin dört adet nokta varsa ve ikili rotalar yapılacaksa, daha önce bahsedilen kombinasyon hesaplama sonucuna göre altı adet rota adayı için kontrol yapılması gerekmektedir. Kümeleme işlemi ile de algoritmalara çok büyük bir performans kazanımı sağlanabilir. Peki, kısıtlama yerine, her bir kısıta uyumlu olan noktaların da belli bir kümeye alınıp çalışmanın hızlandırılması ne kadar mümkündür? Bunu yapabilirsek, her bir kısıt çalışmadan önce birbirleri ile rotalanması gereken noktaları aynı rotaya eklemleyebilme imkânı oluşacaktır ve kısıt işlemleri çok büyük kontroller yapmamış olacaktır. Bu başarıldığı zaman, hem çözüm uzayı küçültülmüş hem de kısıt işlemlerinden kazanç sağlanmış olunacaktır. Çünkü kontrol edilmemesi gereken ve daha önce kümeleme işlemi olabilecek bir kısıtın her bir rota üstünde denenmesi önlenmiş olacaktır.

Bunu aşağıda anlatacağımız biçimde (yukarda bulunan kümeleme işlemleri olmadan kısıt sağlama algoritmasında değinildiği gibi) bir algoritmaya dökmemiz mümkündür.

Noktaların birbirine yakın olması bir kısıtımız olsun (gerçek uygulamalarda bu kısıt oldukça kullanılan bir yöntemdir). Bununla birlikte Çizelge 4.2'deki bir algoritma yazabiliriz;

Çizelge 4.2: Kümeleme yöntemi ile bir kısıtın azaltılması ve performans artırımı

Adım	**İşlem**
1	Öncelikle tek aşamalı bir kümeleme yapılır.
2	Tek aşamalı bu kümeleme işlemi çoğunlukla nokta / coğrafi sistem yönelimli bir kümeleme olur.
3	Birbirine yakın noktalar aynı küme içerisine alınır.
4	Yukardaki tabloda bulunan algoritma basamakları bu basamaktan itibaren uygulanır.

Kümeleme yöntemi aynı kümeye gelecek rotaların, birbirleri ile uyumlu olması kabulü ile hareket ederek kısıtlarda kontrole ihtiyaç bırakmayacak ve yapılacak işlem sayısı azaltılacaktır. Azalma hem çözüm kümesi açısından alacaktır hem de yapılacak kontroller bakımından alacaktır.

Bu dokümanda nokta yönelimli kümelemeye ek olarak zaman aralığı temelli kümeleme kullanılacaktır. Zaman aralığı, herhangi bir varış noktasının çalışma zaman aralığıdır. Bir varış noktası örneğin sabah altı ile akşam beş arasında çalışıyorsa, zaman aralığı / time Windows olarak 06:00/17:00 alınmaktadır. Bu varış noktasına ulaştırılacak yüklemeler de bu zaman aralığı içerisinde ulaştırılmalıdır.

Karşılaşılan tüm rotalama yöntemleri zaman aralığı için iyileştirme yapma yerine uygunluk için çabalamaktadırlar. Kombinasyon sırasında her bir aday rota için zaman aralığı bulma üstüne çalışmaktadırlar. Zaman aralığı için kümeleme yapma ve ardından rota oluşturma veya bunun iyileştirmesi az çalışılan bir konu olmaktadır.

Ayrıca çok büyük sistemler (örneğin GittiGidiyor.com) günlük milyonlara varan rotalama işlemi yapabilmektedir. Bunun sağlıklı yatay bir büyüme ile sağlanabilmesi de gerekmektedir. Böylece yapacağımız öneri ile çok büyük veri

girdileri olan sistemlerde bunun daha iyi ve hızlı sonuçlar verebileceği düşünülmektedir.

Her bir kısıtın bir kümeleme işlemi şeklinde ifade edilmesi, asimptotik olarak hem performansa etki edecektir hem de kısıt çalıştırma işlemlerinin yapılmaması ile çakışma zamanına katkı sağlayacaktır. Çalışmada sadece zaman aralığı kısıtlarını kaldırarak bir kümeleme işlemine çevirme önerisi yapmaktayız. Ancak zaman aralığı kısıtları kaldırılması ve ayrıca diğer kısıtların yine kümeleme işlemi şekline getirilmesi uygulanmamıştır. Bunu yapabilmek başka bir çalışmaya konu olmak üzere bırakılmıştır.

5 KULLANILAN YÖNTEM

5.1 Metot

Rotalama için kullanılan veri setimiz dört boyutlu bir matris olarak düşünülmüştür. Kümeleme yaparken bu matris üstünden işlem yapılmaktadır. Boyutlar sırası ile şu şekildedir;

1. x,
2. y,
3. başlangıç-za,
4. bitiş-za

Her bir boyutun anlam olarak açıklaması ise aşağıdaki şekildedir.

- "x" ve "y" boyutları coğrafi koordinatlardır.
- "başlangıç-za" ve "bitiş-za" ise zaman aralığını ifade etmektedir.

Kabul edilen varsayım ise bu şekilde kümelenen veri setinin uygun kümeleme yapabileceğidir. Böylece zaman aralığı olarak birbirine uyumsuz olan varış noktaları aynı küme de bulunmayacaktır. Başka bir varsayım ise, aynı küme içerisinde bulunan noktalar, aynı zamanda birbirlerine yakın olacak noktalar olmasıdır.

Belirtilmesi gereken diğer bir nokta ise, bu çalışmanın tek bir boyut sıralaması ile değil de, bir den çok sıralama kombinasyonu ile denenmiş olduğudur. Yani ilk denemeler

1. x,
2. y,
3. başlangıç-za,
4. bitiş-za

sıralaması ile yapılmış, ardından, zaman aralıklarının boyut sırasına göre etkisinin görülebilmesi için de ilk boyuta zamanlama kısımları eklenmiş ve denenmiştir.

1. başlangıç-za,
2. bitiş-za
3. x,
4. y,

şeklindeki sıralama yapılması sonucu çıkan sonuçların uygunluğu takip edilmiştir. Sonuç bölümünde belirtildiği gibi en iyi rotalama çıktıları böyle kombinasyon denemelerinin sonucudur. Ancak bulduğumuz en iyi sonuçları veren boyut sıralamasının ilk belirtilen

1. x,
2. y,
3. başlangıç-za,
4. bitiş-za

sıralaması olduğu görülmüştür. Bu sıralama en fazla sayıda sonucun iyi bir çıktı vermesini sağlamıştır. Diğer sıralamalar ise bazı özel veri girdilerinde işe yaramış ama genel olarak, çoğu veri girdilerinde en iyi sonucu verdiğini belirtmek yanlış olacaktır. Yukarıda belirtilen boyut sıralamaları harici diğer tüm sıralamalar da denemiş ancak daha iyi bir sonuç bulunamamıştır.

Kümeleme algoritması olarak da K-Ortalama kullanılmıştır. Geleneksel iki boyutlu değil de dört boyutlu bir kümeleme problemi olarak ele alınmıştır. Ayrıca kümeleme algoritmamız büyük veri ortamında çalışan bir yapıdadır. K-Ortalama algoritması ise mllib içerisinde gerçekleştirilen *org.apache.spark.mllib.clustering.KMeans* sınıfıdır.

Bu dokuman başlangıçta sistemin çalışması için gereken kurulumlar ve ayarlamaların anlatılması ile başlayacaktır. Ardından çözüm yönetimimizin ayrıntılı bir şekilde açıklanması ile devam edecektir. Daha sonra bulunan bazı sonuçlar gösterilecektir. En sonda olarak da ilerde yapılabilecekler üstünde bazı açıklamalar yapılacaktır.

5.2 MLLib K-Ortalama Algoritması

K-Ortalama algoritması bir kümeleme algoritmasıdır. Kümeleme algoritmaları da denetimsiz öğrenme algoritmalarındandır. (Apache Yazılım Vakfı, 2016). Belirli bazı benzerlik ölçütlerine bağlı olarak kümeleme işlemleri yapmaktadır.

K-Ortalama en çok kullanılan kümeleme algoritmalarından birisidir. Girdi veri noktalarını daha önce belirtilen bir sayıda kümelere böler.

MLLib içerisinde kullanılan algoritma, "k-means++" algoritmasının bir parallel olmak üzere oluşturulmuş bir çeşidi olan "k-means||" algoritmasıdır (Bahmani, 2012). MLLib içerisinde bulunan gerçekleştirilme şeklinde ise aşağıdaki parametreler bulunmaktadır;

- k:istenilen küme adedi
- enFazlaDöngü: algoritma içerisinde kaç adet döngü istenildiği
- başlamaŞekli: rastgele olarak seçilmiş noktalarla mı yoksa "k-means||" ile belirtilmiş biçimde mi başlanmış olması gerektiği.
- koşmaSayısı: Algoritmanın kaç adet koşması gerektiği.
- başlangıçBasamakları: "k-means||" algoritmasında kaç adet başlangıç basamağı olması gerektiği.
- Epsilon/değişim: k-ortalamanın artık yeterinde koştuğunu belirten değişim miktarı
- ilkModel: seçimlik bir girdi parametredir ve algoritmaya sağlanırsa tek bir koşma ile sonuca gidilir ve ilk en iyi merkezi noktalar bulma işlemi yapılmaz.

5.2.1 K-Ortalama Algoritmasının Çalışma Mantığı

$X=\{x_1,\ldots,x_n\}$ kümesinin d-boyutlu bir Öklid uzayı olduğu düşünülsün. ***k*** ise küme sayısını belirtir sıfırdan büyük bir tam sayıdır.

$$||x_i - x_j||$$

ise x_i ve x_j arasında tanımlanan Öklid uzaklığını belirtir. Bir x noktası ve $Y \subseteq X$ alt kümesi için uzaklık

$$\text{enküçük}_{y \in Y} ||x - y||.$$

şeklinde belirtilir. $Y \subseteq X$ alt kümesi için kütle merkezi aşağıdaki gibi belirtilir;

$$kitlemerkezi(Y) = \frac{1}{Y}\sum_{y \in Y} y$$

Asıl K-ortalama algoritması hesaplama basamakları ise şu şeklidedir;

- Rastgele seçilen *k* adet merkez nokta bulunur

- Her bir döngü bu merkezleri baz alarak türetilir.
- Yeni eklenen her bir nokta ağırlık merkezini değiştirir böylece yeni ağırlık merkezi bulunur
- Sitem belirli bir döngü ardından artık çok değişmemeye başlar ve böylece sonuç bulunur.

Bu şekilde çalışma aslında yerel iyi bir sonuç bulma için uygulanan bir yöntemdir ve piyasada en çok kullanılan yapılardan birisidir (Bahmani, 2012).

5.2.2 K-means++ Algoritması

İlk hazırlama kısmında bulunan başlangıç merkezlerin bulunması için k-ortalama algoritması rastgele merkezler seçme şeklindedir. "K-means++" algoritmasında bu özelleştirilmiştir (Arthur, 2007). Burada asıl amaç merkezleri rastgele değil de, daha önce yapılan seçimlere bağlı olarak seçmektir. Böylece daha iyi sonuçlar bulabilmişlerdir. Algoritma ilk önce var olan noktalardan bir tanesini ilk merkez olarak seçer. Ardından istenilen merkez nokta sayısına ulaşılıncaya kadar diğer noktalardan diğer merkezleri seçmeye çalışır. Bunu yaparken de ilk merkezle yeni bulunmak istenen merkezler arasında belirli bir formüle göre uzaklık hesabı yapar. En iyi sonucu veren nokta yeni merkez nokta olarak kabul edilir.

5.2.3 K-means|| Algoritması

Aynı şekilde K-means|| algoritması da ilk çalışan merkez bulma işlemleri sırasında yapılan bir iyileştirme işleminden ibarettir. Bunu yaparken olasılık hesapları yaparak daha iyi sonuçlar bulmayı amaçlar. Ancak bu yöntemin bir dezavantajı bulunmaktadır. O da çıkan küme sayısı istenilen küme sayısından fazla olabilmektedir. Bu nedenle fazlalık noktaları oluşan diğer kümelere dağıtmaya çalışır. Bu algoritmanın bir özelliği de parallel çalışabilmesidir. Bu nedenle Spark içerisinde kullanılma imkânı oluşmuştur. Hangi algoritma olursa olsun, Hadoop veya alt projelerinde kullanılmak isteniyorsa, ya parallel olarak çalışmalıdır veya parallel forma dönüşmesi sağlanmalıdır. Bu nedenle kullanılan k-ortalama algoritmasının bu türevi uygun olmuştur.

5.3 Hazırlıklar

Tez gerçekleştirmesi için yapılan kurulumların ve uygulama ayarlamalarını bu kısımda anlatılacaktır. Hortonworks Veri Platformu ile iki türlü Hadoop kurulumu gerçekleştirme yöntemi bulunmaktadır;

1. Kurulum dosyaları kullanılarak Windows ve Linux işletim sistemleri üzerinde
2. Ambari kullanılarak Linux ve benzeri işletim sistemleri üzerinde.

Kurulum sırasında yapılan tüm işlemler ve yapılması gerekenler EK-1 içerisinde anlatılmıştır.

Windows işletim sistemi üzerindeki kurulum işlemleri ayrıntılı olarak, Ambari kullanımı ile kurulum işlemleri özet olarak anlatılmıştır. İki sistemin anlatılmasının nedeni ise, bazı sistem kurulumlarında iki faklı türde işletim sistemi bulunabilmesidir. İki farklı işletim sistemi barındıran bir Hadoop kurulumu yapılmadı, ancak Spark kullanılabilecek makineler üzerinde hem Windows hem de Linux tabanlı makinelerin kullanımı yapılmıştır.

Kurulum olarak kullanılan iki farklı yöntem ile denemiştir. İlk yöntemde üç adet windows işletim sistemi barındıran makine bulunmakta ve ayrıca Spark ile kullanılabilecek iki adet Linux makinası bulunmakta idi. Bu denemede, Hadoop kurlumu windows makineler üzerinde yapılmış, HDFS sistemi bu makinalar üzerinde ayarlandı. Kullanılan girdi veriler bu sistem üzerine kopyalandı. Ardından Spark kurulumu sonrası bir Windows makinesi Spark efendisi olarak tanıtılıp diğer dört makine Spark köleleri olarak tasarlandı. Çalışma için yapılan koşma işlemleri bu Spark kümesi üzerinde denendi. İkinci yöntem de ise, tüm makinalar Linux olarak tasarlandı ve Spark için de kurulum yine bu makinalar üzerinde yapıldı. Tez için yapılan koşturma işlemleri bu makineler üzerinde denendi.

Tüm bu kurulum ve denemelerden alınan dersler şu şekilde belirtilebilir;

- Windows makinalarında kurulum Ambari ile yapılamadığından daha zor olmaktadır ve vakit almaktadır.
- Linux makinalarında ilk ayarlamalar (ssh ayarlamaları vs.) sonrası kurulum çok kolayca bitirilmektedir.

- Hadoop kümesi olarak kullanılan makineler tek bir işletim sistemi barındırırsa yönetimi çok kolay olmakta, ancak öbür türlü yönetim ve yeni makine ekleme işlemleri zorlaşmaktadır.
- Linux makineler üzerinde yapılan koşma işlemlerinin daha hızlı olabildiği gözlenmiştir.
- Birkaç yıl öncesine göre, yine de kurulum işlemleri kolaylaşmış ve yönetim daha iyi yapılır hale gelmiştir.
- Spark'ın herhangi bir işletim sisteminde benzersiz çalışması bu çalışma için çok büyük bir artı olmuş ve iyileştirme çalışmamızın daha hızlı bitmesini sağlamıştır.

Yapılan başka bir hazırlık işlemi de, her bir kurulumun ardından gerçekleştirilen konfigürasyon değişiklikleridir. Bu kadar çok sayıda alt araçla sahip bir sistemin elbette ki konfigürasyon işlemleri çok vakit alabilmektedir. Özellikle kullanılan portların düzenlenmesi, her bir makinada aynı portların kullanılması kısmı işlemleri zorlaştırmaktadır. Bu işlem ardından varılan ayrı bir sonuç ise, Hadoop ve alt araçlarının kurulduğu makinelerde Hadoop harici başka hiçbir kurulumun yapılmaması ve makinelerin sadece hesaplama için yapılan bu kurulum tarafından kullanılmasıdır. Öbür türlü başka uygulamaların Hadoop ekosistemini nasıl etkileyebileceği çok belirgin olmamaktadır. Bu nedenle Hadoop ekosistemi harici kurulumlara ihtiyaç bulunursa, bunun ek başka makinelerle yapılması daha uygun olmaktadır. Örneğin ilk denemelerimizde, prototip geliştirme amacı ile kullanılan Prediction.IO kurulumu aynı windows makinalara yapılmıştır. Ardından belirli bir süre sonra windows makinalarının yavaşladığı ve hatalar aldığı görülmüştü.

Alınan başka bir ders olarak da, kurulumların el ile değil de, açık kaynak konfigürasyon araçları ile yapılabiliyor olmasıdır. Bunlardan Chef/Puppet gurubu ile kurulumun çok daha kolay olabileceği anlaşıldı. Ancak bu çalışmada bu gibi araçları kullanılmamıştır.

5.4 Gerçekleştirme

Bu tezde geliştirme dili olarak Scala, uygulama geliştirme ortamı olarak da IntellyJ IDEA kullanılmıştır. Ek kütüphaneler olarak da Mllib, Spark ile denemeler yapılmıştır.

Programlama dili olarak Scala seçilmesinin sebebi Spark yazılımının Scala dili ile yazılmış olmasıdır. Ayrıca Scala dili fonksiyonel programlama için çok güzel yapılar sunmaktadır. Scala ayrıca çok hızı bir geliştirme döngüsüne sahip ve bu yapısı sebebi ile de Java programlama dilinde bulunmayan birçok yeni teknolojik özelliklere sahiptir. Scala programlama dilinin bir başka özelliği de Java ile doğal bir şekilde konuşabiliyor olmasıdır. Scala'da yazılan tüm kütüphaneler Java'da da kullanılabilmektedir. Ayrıca Java'da yazılan tüm kütüphaneler de Scala'da kullanılabilmektedir. Bu da Java dili ile yazılmış Hadoop ve ekosistemine bağlantı yaparken çok büyük faydalar sağlamaktadır.

Geliştirme ortamı olarak da IDEA kullanılmasının nedeni ise, Java ve benzeri ortamlara sunmuş bulunduğu destektir. IDEA ayrıca Eclipse ve benzeri geliştirme ortamlarından daha hızlı bir şekilde geliştirilmekte ayrıca komünitede oldukça fazla yer almaya başlamıştır. Örneğin Android geliştirme ortamı IDEA sistemine geçirilmiş ve daha önceki geliştirme ortamları terk edilmeye başlanmıştır. Bu tek başına IDEA geliştirme ortamının olgunluğunu göstermektedir.

5.4.1 Veri Kümesi

Uygulamamız denenirken Gehring&Homberger veri setinde binlik kısımlar denenmiştir. Bu data set

https://www.sintef.no/globalassets/project/top/vrptw/homberger/1000/homberger_1000_customer_instances.zip

sitesinden indirilebilir.

Kullanılan veri kümesinin bir özelliği de, literatürde en çok kullanılan rotalama veri kümelerinden birisi olan Solomon veri kümesinden geliştirilmiş olmasıdır (Solomon, 254-265.). Bunu yaparken de, nokta sayısı ve taşınan siparişler belirli bir algoritmaya göre çoğaltılmış ve orijinal veri kümesine uygun yapı hazırlanmıştır.

Örnek bir rotalama nesnesi aşağıda belirtildiği şekildedir;

<Veri İsmi>

<Boş Satır>

VEHICLE

NUMBER CAPACITY

K Q

< Boş Satır>

CUSTOMER

CUST NO. XCOORD. YCOORD. DEMAND READY TIME DUE DATE SERVICE TIME

< Boş Satır >

0 x0 y1 q0 e0 l0 s0

1 x1 y2 q1 e1 l1 s1

... ...

100 x100 y100 q100 e100 l100 s100

Her bir veri kümesinde sıfır numaralı müşteri merkezi depoyu belirtmektedir. Her bir eri kümesinde tek bir depo bulunmaktadır. Yükler bu depodan alınıp diğer numaralara sahip müşterilere ulaştırılmaktadır. Ayrıca araç olarak kullanılabilecek miktar belirtilmekte ancak bu çalışmada araç miktarının sonsuz sayıda olduğu düşünülmüştür. Araç sayısına ek olarak belirtilen başka bir değer de araçların taşıma kapasitesidir. Her bir araç bu kapasiteden daha fazla yük taşıyamamaktadır. Araç belirteçlerinden sonra gösterilen satırlarda ise müşteri ve gitmesi gereken yükler belirtilmektedir. Her bir yükün depodan çıkabileceği en erken zaman ve hedef noktaya vara bileceği en erken zaman belirtilmektedir. Tabi olarak da yükün miktarı da aynı satır içerisinde bulunmaktadır. Bu veri kümelerinde işlemleri kolaylaştıran ve ayrıca ilginç olan bir nokta ise merkezi depo ile hedef noktanın uzaklığının süre ile aynı değere sahip olmasıdır. Bu her satırda belirtilen x ve y koordinatlarından elde edilen kuş uçuşu mesafe ile bu iki nokta arası taşıma zamanının aynı olacağı anlamına gelmektedir.

Her Bir Veri Kümesi için Modellemenin Yapılması

Rotalama işlemi başlamadan ve henüz kümeleme işlemi çalışmadan önce, kümeleme için kullanılacak modellemenin yapılması gerekmektedir. Bu işlem mllib içerisinde yapılması gereken bir işlemdir ve modelleme ardından her bir noktanın hangi kümeye ait olabileceği sorgulanabilmektedir. Modelleme sırasında kullanılan değişkenler aşağıdaki gibi düzenlenebilmiştir;

```
val numberOfCenters = 4
val numberOfIterations = 5000
```

İki ayrı kümeleme işlemi uygulanılandan, her bir kümeleme aşamasında bu değerler ayarlanmalıdır. Örneğin yukardaki gösterim ilk aşama olarak daha çok kullanılan nokta yönelimli kümeleme için denenmiştir. Bu ilk kümeleme işlemi sonrası elimizde bulunacak dört kümenin her birisi daha sonra yine kendi içerisinde zaman aralıklı bir kümeleme işlemine uğratılacaksa, bunun için de bir düzenleme yapmak gerekmektedir. En çok kullanılan düzenleme de yine yukardaki gösterim ile aynı olmuştur. Daha büyük değerler daha iyi sonuçları her zaman vermemektedir. Ayrıca modelleme için de beş bin adet çalışma sayısı belirlenmiştir.

5.4.2 Zaman Aralıkları

Modelleme için verilen bir adet girdi şu şekildedir;

```
c1_10_1

VEHICLE
NUMBER                                                    CAPACITY
 250                                                           200

CUSTOMER
CUST NO.  XCOORD.    YCOORD.    DEMAND   READY TIME  DUE DATE
SERVICE                                                   TIME

   0       250          250          0          0        1824          0
   1       387          297         10        200         270         90
```

```
2        5        297        10        955        1017        90
3  355   177   20   194   245   90
```

bu dosyada, örneğin, 100 adet iş bulunmaktadır (yer nedeni ile tüm işler gösterilmemiştir). İlk kolonda olan rakamlar müşteri veya uğrama noktaları belirteçleridir. İkinci ve üçüncü kolondakiler ise nokta bilgileridir (coğrafi koordinat). Beşinci ve altıncı kolondakiler de zaman aralıklarını belirtmektedir.

```
1  387   297   10   200   270   90
```

için belirtmeye çalışırsak, belirtilen yük;

- (387,387) koordinatında bulunan
- 1 numaraları müşteriye gidecektir
- Taşınan yük 10 kadardır
- Merkezi depodan er erken 200 dolaylarında çıkacak
- Ve en geç 270 dolaylarında varmak zorundadır.
- Hedef noktaya varıldığında ise, yükün boşaltma işlemi 90 birim kadar sürecektir.

Her bir dosya bu şekilde anlaşılıp iyileştirme işlemine yüklenmelidir.

5.4.3 Modelleme

Her bir dosya modelleme işlemine hemen uğratılamadığından bazı çevrimler yapmak gerekmektedir. Dosyadan okunan

```
250       250        0      1824
387       297       200       270
5       297       955      1017
355       177       194       245
```

şeklindeki bir satırı, vektör yapısına çevrim için aşağıdaki kod kullanılmaktadır.

```
sc.textFile(file)
  .zipWithIndex()
    .filter(_._2>9)
    .map(_._1)
    .map(_.replaceAllLiterally("  "," "))
  .filter(_!=null)
  .filter(_.trim.length>0)
  .map { line =>
    line.split(separator).filter(_.trim.length>0)
      .map {
        field => field.toDouble
      }.toSeq.toArray
  }
.drop(1).take(2).union(line.drop(4).take(2))
```

Yukarda bahsedilen vektörler oluştuktan sonra artık eğitim (training) başlatılabilir. Bunun için de aşağıdaki kod kullanılmıştır.

```
val kMeansI = new KMeans()
// Setting the parameters
kMeansI.setK(options.numberOfCenters)
kMeansI.setMaxIterations(options.numberOfIterations)
// Return the KMeansModel which we get after running the KMeans
```

```
// algorithm on the data gathered by the DataSource component
val model = kMeansI.run(training)
val toWrite = new File(options.TXT_INSTANCE_FOLDER + "models/" +
file.getName +"/");
toWrite.mkdirs()
model.save(sc,toWrite.getAbsolutePath +(new Date()).getTime)
```

bu işlemden sonra eğitim bitmiş ve modelimiz dosya olarak kaydedilmiştir. Ancak bu ilk aşama kümeleme işlemidir. Çıkan her ilk küme, ardından zaman aralıklı kümeleme işlemine benzer bir şekilde uğratılmaktadır ve çıktı olarak her bir ikincil küme yeni model dosyalarına sahip olmaktadır.

5.4.4 Kümeleme

Modelleme işleminden sonra, kaydedilen her bir modeli kullanarak, çıktı olarak kümelenmiş noktaları almamız gerekmektedir. Oluşturulan model kullanılarak kullanmamız gereken her bir noktanın kümesini almak için aşağıdaki kod kullanılmıştır;

```
val groups = model.predict(training)
 .zipWithIndex()
 .map {
  line =>
   (line._2, line._1)
 }
 .join(trainingIndexes)
 .map {
  line =>
   (line._2._1, line._2._2.apply(0))
 }
 .groupBy(t => t._1);
```

Bu işlem ardından elimizde her bir küme içinde bulunan noktalar bulunmaktadır. Aynı şekilde, ikincil kümeleme noktaları da bulunmaktadır.

5.4.5 Rotalama

Bulunan tüm kümeler ayrı bir rotalama dosyası olarak kaydedilmekte ve rotalamaya uğratılmaktadır. Rotalama dosyası formatı, orijinal girdi veri formatı ile aynıdır. Tek farkı ise içerisinde aynı kümelere ait noktaların bulunmasıdır. Her bir dosyanın (kümenin) rotalanması bittiğinde ise tüm kümeler birleştirilip tek bir sonuç olarak kaydedilmektedir. Yani elimizde modelleme ve kümeleme sonrası örneğin on altı adet küme oluşmuş olsun. Bunlar ayrı ayrı rotalanmakta ve sonuç ayrı bir rotalama dosyası olarak kaydedilmektedir. Bir rotalama dosyası çok basit bir şekilde, her bir satırda bir rota barındıran bir dosyadır. Dosyanın tüm rotaları birleştiğinde küme için yapılan planlamayı belirtmektedir.

Rotalama kütüphanesi olarak (Schröder, 2016) uygulaması kullanılmıştır.

Tüm bu işlemler otuz dakikadan az bir sürede bitmektedir. Bu süre tek bir dosya işlem süresi değil tüm setin rotalanması için geçen süredir. Ancak oluşacak dosya sayısı ve parametrelere göre bu süre değişiklik gösterebilmektedir. Örneğin küme sayısı azaldıkça bu süre artacak, küme sayısı arttıkça bu süre azalacaktır.

Genel olarak işlem şu şekilde çalışmaktadır;

```
val files = RouterUtil.getListOfFiles(options.TXT_INSTANCE_FOLDER)

for(file<-files){
  routeOne( file)
}
```

Yukarda belirtilen routeOne fonksiyonu ise şu şekildedir;

```
def routeOne(file: File)= {
  val files =
RouterUtil.getListOfFiles(GHSCExporter.getLatestExportFolder(file).getAbsolutePa
th)
  var totalCost = 0.0
  var totalJob = 0.0
```

```
 var lstSols = new ParHashMap[VehicleRoutingProblemSolution, Boolean]
 files.par.foreach { f =>
  val solution = route(f, null)
  totalCost += solution.getCost
  totalJob += solution.getUnassignedJobs.size()
  lstSols +=(solution, true)
 }

 println("solution ")
 val routes = new util.ArrayList[VehicleRoute]()
 val uJobs = new util.ArrayList[Job]()

 val alls = lstSols.map(_._1).toSeq.toArray
 for (s <- alls) {
  val sol = s.asInstanceOf[VehicleRoutingProblemSolution]
  RouterUtil.printSol(sol)
  uJobs.addAll(sol.getUnassignedJobs)
  for (r <- sol.getRoutes.toArray()) {
   val route = r.asInstanceOf[VehicleRoute]
   routes.add(route)
  }
 }

 println("Initial Total Cost : " + totalCost)
 println("Initial Total Unassgned Jobs : " + totalJob)

 val lastSol = new VehicleRoutingProblemSolution(routes,uJobs,totalCost)
 var sF = new File(options.TXT_INSTANCE_FOLDER + "schedules/" +
file.getName + "/" );
 sF.mkdirs()
 sF = new File(sF.getAbsolutePath + "/" + file.getName+ "_" + totalCost + "km_" +
totalJob + "uj" +  + (new Date()).getTime + ".sc");
```

```
RouterUtil.printSol(lastSol,sF)
```

6 SONUÇLAR VE TARTIŞMA

Sonuçlarımız Çizelge 6.1'de belirtilmiştir. Tüm çalışma 2.39 GHzx4 işlemcili, 30GB RAM barındıran üç adet Linux makina veya aynı özellikte beş adet Linux + Windows makine üstünde koşturulmuştur. Tüm işlemin bitmesi 30 dakikadan az sürmüştür. Bulunan sonuçlar bu kısa zaman için uygun olan rakamlardır. Bazıları da (kırmızı renkte gösterilenler) literatürde bulunan en iyi performanslardan daha iyi sonuçları göstermektedir.

Çizelge 6.1: Bulunan sonuçlar

Veri	**BEİ Araç**	**BEİ KM**	**BT Araç**	**BT KM**
c1_10_1	100	42478,95	101	42835,7099
c1_10_10	90	39923,41	100	42661,56006
c1_10_2	90	42247,05	102	43258,73915
c1_10_3	90	40101,36	101	42927,31214
c1_10_4	90	39468,6	100	42409,74766
c1_10_5i	100	42469,18	100	42555,15005
c1_10_6	99	43830,21	102	43319,73126
c1_10_7	97	43453,92	101	42781,84419
c1_10_8	92	44092,74	102	43309,19313
c1_10_9	90	40546,6	100	42478,41052
c2_10_1	30	16879,24	37	18753,31485
c2_10_10	28	15944,72	36	18215,25026
c2_10_2	29	17126,39	35	18627,98076
c2_10_3	28	16884,08	36	18639,43735

Çizelge 6.1: (devam) Bulunan sonuçlar

c2_10_4	28	15656,75	36	18177,10679
c2_10_5	30	16561,29	36	19010,09032
c2_10_6	29	16920,33	37	19252,98971
c2_10_7	29	17882,42	38	19732,6096
c2_10_8	28	16577,32	37	18678,23437
c2_10_9	29	16370,44	36	19150,29173
r1_10_1	100	53501,39	124	63020,59335
r1_10_10	91	48294,71	97	51895,70664
r1_10_2	91	49105,21	110	58315,33837
r1_10_3	91	45235,85	117	52298,27706
r1_10_4	91	42787,19	111	49160,87061
r1_10_5	91	51830,36	99	56066,38438
r1_10_6	91	47832,22	110	54454,1879
r1_10_7	91	44435,5	106	50649,2103
r1_10_8	91	42485,38	103	47271,48076
r1_10_9	91	50490,49	99	54075,82481
r2_10_1	19	42188,86	33	39951,00347
r2_10_10	19	30215,24	26	32423,1039
r2_10_2	19	33459,32	31	33921,62639
r2_10_3	19	24938,95	25	26466,57965
r2_10_4	19	17880,11	23	19735,14182
r2_10_5	19	36232,18	31	37235,98753
r2_10_6	19	30073,6	25	31491,4107
r2_10_7	19	23253,89	24	25593,92047

Çizelge 6.1: (devam) Bulunan sonuçlar

r2_10_8	19	17495,51	22	19371,92529
r2_10_9	19	33002,36	27	34981,0471
rc1_10_1	90	46272,07	103	52052,29685
rc1_10_10	90	43896,78	95	47536,47268
rc1_10_2	90	44129,42	98	51209,81466
rc1_10_3	90	42487,54	103	47839,52726
rc1_10_4	90	41613,58	105	47362,01087
rc1_10_5	90	45564,81	100	50475,62129
rc1_10_6	90	45303,67	99	50126,94452
rc1_10_7	90	44903,8	97	48827,21801
rc1_10_8	90	44366,01	98	49128,56533
rc1_10_9	90	44280,84	97	48478,63052
rc2_10_1	20	30278,5	34	30348,38089
rc2_10_10	18	21910,33	21	23810,70587
rc2_10_2	18	26327,92	31	26444,65828
rc2_10_3	18	20043,04	25	21447,9269
rc2_10_4	18	15741,56	22	17697,13047
rc2_10_5	18	27140,77	30	28226,93507
rc2_10_6	18	26797,76	33	28238,15741
rc2_10_7	18	25112,77	28	26910,30293
rc2_10_8	18	23709,29	27	25631,29288
rc2_10_9	18	23028,1	25	24845,67377

BEİ: Bilinen en iyi

BT: Bu tez

Bu tabloda dikkat çeken kısım algoritmanın tüm kümede iyi bir sonuç bulamaması ama bazılarında en iyi sonucu bulmasıdır. Bu bize biraz iyileştirme ile çok daha iyi bir sonucun bulunabileceğini göstermektedir.

KAYNAKLAR

Sorting 1PB with MapReduce. (2008, 11 21). Google: https://googleblog.blogspot.com.tr/2008/11/sorting-1pb-with-mapreduce.html adresinden alınmıştır

***Cloudera Customers*.** (2016, 11 21). Cloudera: http://www.cloudera.com/customers.html adresinden alınmıştır

Dağıtımlar ve Ticari Destek . (2016, 11 21). Apache Yazılım Vakfı: https://wiki.apache.org/hadoop/Distributions%20and%20Commercial%20Support adresinden alınmıştır

Almasi, G. S. (1989). *Highly parallel computing.* Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc.

Apache Yazılım Vakfı (2016, 11 06). *Welcome to Apache Hadoop!* http://hadoop.apache.org/ adresinden alınmıştır

Apache Yazılım Vakfı, Spark Projesi (2016, 12 4). *Clustering - RDD-based API.* Apache Spark 2.0.2 Documentation: http://spark.apache.org/docs/latest/mllib-clustering.html#k-means adresinden alınmıştır

Arthur, D., & Vassilvitskii, S. (2007, January). k-means++: The advantages of careful seeding. In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms (pp. 1027-1035). Society for Industrial and Applied Mathematics.

Bahmani, B. M. (2012). Scalable k-means++. *Proceedings of the VLDB Endowment, 5(7)*, 622-633.

Baldacci, R. B. (2010). Some applications of the generalized vehicle routing problem. *Journal of the Operational Research Society, 61(7)*, 1072-1077.

Beasley, J. E. (1983). Route first—cluster second methods for vehicle routing. *Omega, 11(4)*, s. 403-408.

Boyd, D., & Crawford, K. (2012). Critical questions for big data: Provocations for a cultural, technological, and scholarly phenomenon. *Information, communication & society* (s. 662-679). içinde

Bräysy, O., & Gendreau, M. (2005). Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. Transportation science, 39(1), 104-118.

Cordeau, J. F. (2003). The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies, 1(2)*, 89-101.

Crainic, T. G. (2010). Two-echelon vehicle routing problem: a satellite location analysis. *Procedia-Social and Behavioral Sciences, 2(3)*, 5944-5955.

Crispim, J., & Brandão, J. (2005). Metaheuristics applied to mixed and simultaneous extensions of vehicle routing problems with backhauls. Journal of the Operational Research Society, 56(11), 1296-1302. Culler, D. E. (1999).

Parallel computer architecture: a hardware/software approach. Gulf Professional Publishing.

Czech Z J, C. P. (2002). A parallel simulated annealing for the vehicle routing problem with time windows. *Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, (s. 376-383). Canary Islands, Spain.

Çatay, B. (2010). A new saving-based ant algorithm for the vehicle routing problem with simultaneous pickup and delivery. *Expert Systems with Applications, 37(10)*, 6809-6817.

Dantzig G B, R. J. (1959). The Truck Dispatching Problem. *Management Science, 6(1)*, 80–91.

Dean, J., & Ghemawat, S. (2008). *MapReduce: simplified data processing on large clusters*. http://research.google.com/archive/mapreduce.html adresinden alınmıştır

Dondo, R., & Cerdá, J. (2007). A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. European Journal of Operational Research, 176(3), 1478-1507.

Eiselt, H. A. (1995). Arc routing problems, part II: The rural postman problem. *Operations research, 43(3)*, 399-414.

Ghemawat, S. G. (2003). The Google file system. *ACM SIGOPS operating systems review, 37*(5), 29-43.

Gonzalez-Feliu, J. P. (2008). *The two-echelon capacitated vehicle routing problem.* https://halshs.archives-ouvertes.fr/halshs-00879447/ adresinden alınmıştır

Granada, M. (2016). LITERATURE REVIEW ON THE VEHICLE ROUTING PROBLEM IN THE GREEN TRANSPORTATION CONTEXT. *revista. luna. azúl, 42*, 362-387.

Hoffman, A. J. (1986). The traveling salesman problem: a guided tour of combinatorial optimization. J. Wiley & Sons.

Holmes, A. (2012). *Hadoop in practice.* Manning Publications Co.

Jemai, J. Z. (2012, April). An NSGA-II algorithm for the green vehicle routing problem. *In European Conference on Evolutionary Computation in Combinatorial Optimization* (s. 37-48). Springer Berlin Heidelberg.

Kara, I., & Bektas, T. (2003, July). Integer linear programming formulation of the generalized vehicle routing problem. In EURO/INFORMS Joint International Meeting, Istanbul, July (pp. 06-10).

Kumar, S. N. (2012). A survey on the vehicle routing problem and its variants. *Intelligent Information Management, 4(3)*, 66.

Laney, D. (2001). 3D data management: Controlling data volume, velocity and variety. *META Group Research Note, 6*, 70.

Laporte, G. D. (1984). Two exact algorithms for the distance-constrained vehicle routing problem. *Networks, 14(1)*, 161-172.

Li, J. M. (2013). *Prajna: Cloud Service and Interactive Big Data Analytics*.

Lynch, C. (2008). Big data: How do your data grow? *Nature*, 28-29.

Flood, M. M. (1956). The traveling-salesman problem. Operations Research, 4(1), 61-75.

Mashey, J. R. (1997). Big Data and the next wave of infraS-tress. *Computer Science Division Seminar.* University of California, Berkeley.

MIKE CAFARELLA, D. C. (2004, 4). Building Nutch: Open Source. *Magazine Queue - Search Engines Volume 2 Issue 2*, s. 54.

Nagy, G., & Salhi, S. (2007). Location-routing: Issues, models and methods. European Journal of Operational Research, 177(2), 649-672.

O'Malley, O. (2008, 5). *Terabyte sort on apache hadoop.* Yahoo!: http://sortbenchmark. org/Yahoo-Hadoop adresinden alınmıştır

Ralphs, T. K. (2003). On the capacitated vehicle routing problem. *Mathematical programming, 94(2-3)*, 343-359.

Sakalli, A., Yesil, E., Musaoglu, E., Ozturk, C., & Dodurka, M. F. (2013, November). Heuristic bubble algorithm for a linehaul routing problem: An extension of a vehicle routing problem with pickup and delivery. In Computational Intelligence and Informatics (CINTI), 2013 IEEE 14th International Symposium on (pp. 435-439). IEEE.

Savran, A. I. (2014). RouteArt: A new framework for vehicle routing problem with pickup and delivery using heuristic bubble algorithm. *Computational Intelligence and Informatics (CINTI), 2014 IEEE 15th International Symposium on*, 91-96.

Savran, A. I. (2015). An efficient solution to Location-Routing Problems via a two-phase heuristic bubble approach. *Advanced Logistics and Transport (ICALT), 2015 4th International Conference on*, 169-174.

Schröder, S. (2016, 12 04). *jsprit-project*. http://jsprit.github.io/ adresinden alınmıştır

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The hadoop distributed file system mass storage systems and technologies (MSST) 2010. In IEEE 26th Symposium on (p. 10).

Solomon, M. M. (254-265.). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research, 35(2)*, 1987.

Vigo, D. (1996). A heuristic algorithm for the asymmetric capacitated vehicle routing problem. *European Journal of Operational Research, 89(1)*, 108-126.

White, T. (2015). Hadoop: The Definitive Guide 4th Edition.

Yildirim, U. M. (2012). A time-based pheromone approach for the ant system. *Optimization Letters 6.6*, 1081-1099.

Yuce, M. F. (2016). Enhancing heuristic bubble algorithm with simulated annealing. *Cogent Business & Management, 3*(1).

Zambito, L. (2006). The traveling salesman problem: a comprehensive survey. *Project for CSE 4080.* içinde

ÖZGEÇMİŞ

Nisantepe Mah.

Alemdag Emlak Konutlari Blok:B1 - D :48 Cekmekoy

İSTANBUL/ TURKEY

mfyuce@gmail.com

Mehmet YUCE +90 538 609 09 89

1981, Patnos, Turkey

Experiences

Organon Analytics

01.2016-..

Sr. Software Engineer Istanbul

LA Logistics & Associates

12.2013-01.2016

Sr. Software Engineer Istanbul

Logo Business Solutions

05.2013-11.2013 Sr. Software Engineer Kocaeli

Intigral

05.2012-04.2013 Consultant - Systems & Application Development Dubai

PIWORKS

10.2007-03.2012 Sr. Software Engineer San Francisco