KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

YÜKSEK LİSANS TEZİ

HADOOP EŞLE/İNDİRGE MİMARİSİ KULLANILARAK GÖRÜNTÜLERİN EN UZUN ORTAK ALT DİZİ ALGORİTMASI İLE ÖRÜLMESİ

ENES YÜCER

KOCAELİ 2019

KOCAELİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS TEZİ

HADOOP EŞLE/İNDİRGE MİMARİSİ KULLANILARAK GÖRÜNTÜLERİN EN UZUN ORTAK ALT DİZİ ALGORİTMASI İLE ÖRÜLMESİ

ENES YÜCER

Doç.Dr. Ahmet SAYAR		
Danışman,	**Kocaeli Üniversitesi**	.....
Doç.Dr. İbrahim ÖZÇELİK		
Jüri Üyesi,	**Sakarya Üniversitesi**	.....
Dr.Öğr.Üyesi Alev MUTLU		
Jüri Üyesi,	**Kocaeli Üniversitesi**	.....

Tezin Savunulduğu Tarih: 11.02.2019

ÖNSÖZ VE TEŞEKKÜR

Bu tez çalışmasında görüntü örme adımlarının Hadoop Eşle/İndirge programla modeli kullanarak geliştirilmesine yönelik çalışmalar yapılmıştır.

Tez çalışmamda desteğini esirgemeyen, çalışmalarıma yön veren, bana güvenen, rehberlik eden ve yüreklendiren danışmanım Doç.Dr. Ahmet SAYAR'a sonsuz teşekkürlerimi sunarım.

Tez çalışmamın tüm aşamalarında bilgi ve destekleriyle katkıda bulunan hocam Arş.Gör.Dr. Süleyman EKEN'e teşekkür ediyorum.

Eğitim hayatım boyunca manevi desteklerini esirgemeyen aileme ve eşim Elif YÜCER'e teşekkürlerimi sunarım.

Ocak - 2019 Enes YÜCER

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

Kısaltmalar

BK	:	Bakınız
CM	:	Cloudera Manager
CUDA	:	Compute Unified Device Architecture (Hesap Birleşik Aygıt Mimarisi)
GFS	:	Google File System (Google Dosya Sistemi)
GPU	:	Graphics Processing Unit (Grafik İşlemci Ünitesi)
HDFS	:	Hadoop Distributed File System (Hadoop Dağıtık Dosya Sistemi)
HIB	:	Hipi Image Bundel (Hadoop Görüntü İşleme Ara Yüzü Görüntü Yığını)
HIPI	:	Hadoop Image Processing Interface (Hadoop Görüntü İşleme Ara Yüzü)
KK	:	Komşuluk Kodu
LCS	:	Longest Common Substring (En Uzun Ortak Alt Dizi)
MIPr	:	MapReduce Image Processing Framework (Eşle/İndirge Görüntü İşleme Arayüzü)
SIFT	:	Scale Invariant Feature Transform (Ölçek Bağımsız Özellik Dönüşümü)
SURF	:	Speed-up Robust Features (Hızlandırılmış Gürbüz Özellikler)
SVM	:	Support Vector Machine (Destekçi Vektör Makinesi)

HADOOP EŞLE/İNDİRGE MİMARİSİ KULLANILARAK GÖRÜNTÜLERİN EN UZUN ORTAK ALT DİZİ ALGORİTMASI İLE ÖRÜLMESİ

ÖZET

Bu tezde, görüntü örme adımlarının dağıtık bir şekilde en uzun ortak alt dizi algoritması kullanılarak yapılabilmesi için görüntü örme süreçleri incelenmiş ve görüntü örme adımlarının dağıtık veri işleme altyapısı sunan Apache Hadoop ortamlarında çalışabilmesi için yeni çözümler ve algoritmalar geliştirilmiş.

Öncelikle, görüntü örme süreçleri, büyük veri kavramı ve büyük veri işleme kütüphanesi olan Apache Hadoop Framework'ü incelenmiş, görüntülerin metin olarak ifade edilebilmesi Komşuluk Kodu algoritması ve metin olarak ifade edilen resimlerin karşılaştırılması için En Uzun Ortak Alt Dizi (Longest Common Substring-LCS) algoritması geliştirilmiştir. Sonrasında, geliştirilen algoritmaların Apache Hadoop Framework'ünün Eşle/İndirge programlama modeline uygulanabilirliğini göstermek için tek makinede tek thread, çok thread çalışan çözümler geliştirilmiştir. Sonrasında zincirleme üç Eşle/İndirge işinden oluşan Apache Hadoop ortamlarında çalışacak çözüm geliştirilmiştir.

Anahtar Kelimeler: En Uzun Ortak Alt Dizi, Eşle/İndirge, Görüntü Örme, Hadoop.

DISTRIBUTED IMAGE STITCHING WITH LONGEST COMMON SUBSTRING ALGORITHM USİNG HADOOP MAPREDUCE ARCHITECTURE

ABSTRACT

This thesis, image stitching process is examined and analyzed to develop new distributed image stitching algorithm with longest common substring algorithm using hadoop mapreduce architecture.

Firstly, image stitching process, big data and Apache Hadoop framework are examined. New algorithm which is name “neighborhood code generator”, is developed that convert image to textual format. The longest common substring algorithm is developed to compare images expressed as text. Single threaded, multi-threaded solutions are developed on a single machine to demonstrate the applicability of the developed algorithms to the mapreduce programming model of the Apache Hadoop Framework. Afterwards, the solution is developed to work in Apache Hadoop environments, which contain three mapreduce chain jobs.

Keywords: Longest Common Substring, Map/Reduce, Image Stitching, Hadoop.

GİRİŞ

Görüntü örme, iki yada daha fazla sayıda görüntünün içerisinde eşleşen noktaların bulunup birleştirilerek panoramik veya yüksek çözünürlükte görüntüler üretebilmeyi sağlayan, fotoğrafların çoklu şekilde birleştirme işlemine verilen isimdir. Aynı zamanda görüntü mozaikleme olarak da bilinir. Günümüzde görüntü örme işlemini yapan özel yazılımlar olduğu gibi modern dijital kameraların çoğu panorama oluşturma modu içerir.

Görüntü örme, panoramik manzara görüntülerinin oluşturulmasında bilgisayarlarda, askeriyede, tıbbi görüntülemede, yüksek çözünürlüklü fotoğraf mozaiklerinde, uydu fotoğrafçılığında görüntüleri karşılaştırmak ve birleştirmek için kullanılmaktadır.

Görüntü örme, temel görüntü işleme algoritmaları kullanılarak yapılmaktadır. Görüntü işleme algoritmaları kullanılarak geliştirilen çözümler alan tabanlı ve özellik tabanlı olmak üzere temelde iki sınıfı ayrılmışlardır (Bölüm 1.4'de bk.). Alan tabanlı teknikte görüntülere ait pikseller tek tek karşılaştırılmaktadır (Bölüm 1.4.1'e bk.). Özellik tabanlı yaklaşımda ise görüntülerden çıkartılan özellikler arasında ilişkiler üzerinden benzerlik bulunur (Bölüm 1.4.2'ye bk.).

Görüntü örme işleminin başarısı kullanılan algoritma ve tekniğe bağlı olduğu gibi birleştirilmek istenen resimlerinin pozlama açısına, ölçeğine ve resmin boyutuna da bağlıdır.

Gelişen görüntü yakalama (fotoğraf makineleri) teknolojileriyle kaydedilen büyük ölçekli görüntülerin saklanması, işlenmesi mevcut yöntemlerle daha da zorlaşmıştır. Aynı zamanda görüntünün karakteristiğindeki bu değişiklikler görüntülerin saklanmasını ve işlenmesini büyük veri (Bölüm 1.1'e bk.) kapsamında değerlendirilmeye başlanmıştır. Bu süreçte mevcut görüntü örme algoritmalarının büyük veri teknolojileriyle tekrardan geliştirilmesini ve büyük veri teknolojileriyle yeni görüntü örme yaklaşımlarını zorunlu kılmıştır.

Tez kapsamında, büyük veri kavramı, büyük veri teknolojileri, görüntü örme algoritmalarının kırılımları, zorlukları ve darboğazları derinlemesine incelenmiş. Görüntülerin örme adımlarının paralelleştirilmesinde, paralel büyük veri işleme kütüphanelerinden olan hadoop ve eşle/indirge programlama modelinin olumlu performans kazanımları göz önünde bulundurularak eşle/indirge programlama modelinin kullanılarak olumlu performans kazanımları sağlanması hedeflenmiştir Sistemin yapılabilirlik ve ölçeklenebilirliği çeşitli görüntüler üzerinde test edilmiştir.

Bölüm 1.1'de ilk olarak büyük veri kavramının ne olduğu, hangi verilerin büyük veri kapsamında değerlendirildiği ve büyük veri birleşenleriyle ilgi araştırma yapılmış ve açıklanmıştır. Bölüm 1.2'de büyük veri teknolojilerinden Apache Hadoop mimarisi üzerinde durulmuştur. Aynı zamanda Apache Hadoop çatısı altında görüntü işlem amacıyla geliştirilmiş Hadoop Image Processing Interface (HIPI) aracı açıklanmıştır. Bölüm 1.3'te temel görüntü örme yöntemleri üzerinde durulmuştur. Bölüm 2'de, Apache Hadoop mimarisi kullanılarak geliştirilen görüntü işleme çözümleri ele alınmıştır. Bölüm 3'te tez çalışması süresince geliştirilen algoritma açıklanarak probleme çözüm yaklaşımı ortaya konmuştur. Bölüm 4'te geliştirilen algoritmalarla yapılan testlerden bahsedilmiştir. Bölüm 5'te elde edilen sonuçlar yorumlanarak bir sonraki aşamada geliştirilebilecek nitelikler ortaya konmuştur.

1. BÜYÜK VERİ, HADOOP MİMARİSİ VE GÖRÜNTÜ ÖRME YÖNTEMLERİ

Bu bölümde ilk olarak büyük veri kapsamından bahsedilecektir. Sonrasında büyük verileri paralel olarak işlem için geliştirilmiş olan Apache Hadoop mimarisinden bahsedilip Hadoop için özelleşmiş görüntü işleme kütüphanesine değinilecektir. Son olarak görüntü örme yöntemleri üzerinde durulacaktır.

1.1. Büyük Veri

Büyük veri (big data), farklı kaynaklar tarafından üretilen ve geleneksel yöntemlerle işlenemeyen yapısal ve yapısal olmayan verilere verilen isimdir. Büyük veri kavramı teknolojinin ilerlemesi ve internetin gelişmesiyle birlikte firmaların rakiplerle rekabet edebilmek için farklı sistemlerden topladıkları verileri analiz ederek anlamlı veriler elde etmek istemesiyle ortaya çıkmıştır. Google, Facebook, Amazon gibi büyük teknoloji firmaları bu sürece öncelik etmiştirler.

Bir verinin büyük veri sınıfında değerlendirilebilmesi için genel kabul görmüş büyük veri bileşenlerinin en az bir tanesinin özelliklerini içermesi gerekmektedir. Büyük veri Şekil 1.1'de gösterildiği gibi 4 bileşenden oluşmaktadır;

- Verinin büyüklüğü (volume) [1]
- Verinin çeşitliliği (variety) [1]
- Verinin hızı (velocity) [1]
- Verinin doğruluğu (veracity) [1]

Verinin büyüklüğü, verinin miktarını temsil etmektedir. Veri depolarındaki büyük miktarda veriyi ve ölçeklenebilirliği, erişilebilirliği ve yönetilebilirliği verinin büyüklüğü kapsamında değerlendirir.

Verinin çeşitliliği, uygulamalar, sensörler, akıllı telefonlar ve sosyal medya gibi farklı kaynaklardan üretilen resim, video, ses, metin ve kayıt (log) verileri gibi farklı veri türlerini ifade eder. Bu veriler yapısal, yarı yapısal veya yapısal olmayan formdadır.

Verinin hızı, web siteleri, mobil uygulamalar, sosyal medya gibi kaynaklarda verinin ne kadar hızlı üretilip transfer edildiğini ifade eder. Gerçek zamanlı olarak akan yüksek hızlı verinin büyük veri kapsamında işlenmesi şirketle rekabet avantajı sağlar.

Verinin doğruluğu, verinin doğruluğundaki bozuklukları, anormallikleri ve belirsizlik durumlarını ifade eder.

Şekil 1.1. Büyük Veri Bileşenleri [2]

Büyük veri kavramıyla birlikte gelişen veri işleme, depolama teknolojileriyle daha önce işlenemeyen, ölçeklenemeyen ve saklanamayan verilerin işlenmesine olanak sağlamıştır.

1.2. Hadoop Mimarisi

Apache Hadoop, sıradan sunuculardan (commodity hardware) oluşan küme (cluster) üzerinde büyük verileri işlemek amaçlı uygulamaları çalıştıran ve Hadoop Distributed File System (HDFS) olarak adlandırılan bir dağıtık dosya sistemi ile Hadoop Eşle/İndirge özelliklerini bir araya getiren, Java ile geliştirilmiş açık kaynak kodlu kütüphanedir [3].

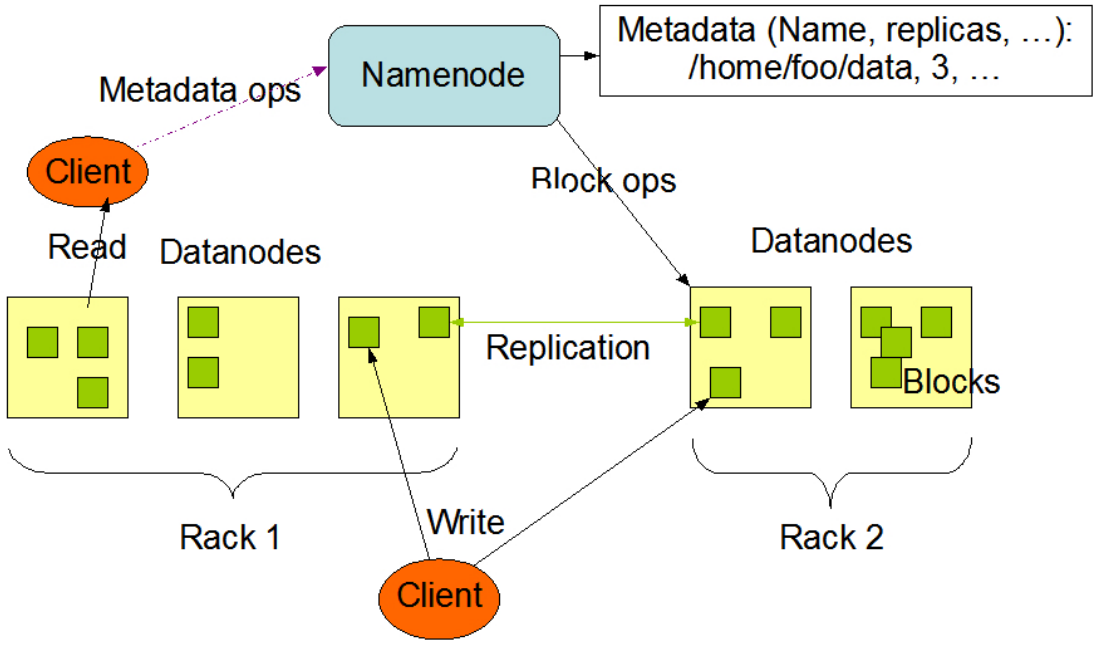
Hadoop, Google Dosya Sistemi (GFS) ve bir sunucu kümesinde depolanan verileri paralel olarak işlemek amacıyla Eşle/İndirge görevlerine ayrılan Eşle/İndirge programlama yaklaşımından ilham almıştır.

1.2.1. Hadoop dağıtık dosya sistemi

Hadoop dağıtık dosyalama sistemi (hadoop distributed file system-HDFS), Hadoop uygulamaları tarafından kullanılan verileri Hadoop sunucu kümesinde yüksek verimli erişimle dağıtık bir şekilde tutan dosyalama sistemidir. Büyük boyutta bir çok dosya bu dosya sisteminde saklanabilir. Dosyalar bloklar halinde birden fazla ve farklı sunucu üzerine dağıtılarak yedeklenir. Veri bloklarının varsayılan boyutu 64MB, kopya sayısı ise 3 olarak ayarlanmıştır. Bu sayede veri kaybı önlenmiş olur. Ayrıca HDFS çok büyük boyutlu dosyalar üzerinde okuma işlemi (streaming) imkanı sağlar, ancak rastlantısal erişim (random access) özelliği bulunmaz. HDFS'te verilerin yönetilmesi ve saklanması için NameNode ve DataNode'lar kullanılır [3-5].

NameNode, veri bloklarının sunucular üzerindeki dağıtımından, yaratılmasından, silinmesinden, bir blokta sorun meydana geldiğinde yeniden oluşturulmasından ve her türlü dosya erişiminden sorumludur. HDFS üzerindeki tüm dosyalar hakkındaki bilgiler (metadata) NameNode tarafından saklanır ve yönetilir. Her kümede yalnızca bir adet NameNode olabilir [6,7].

DataNode, veri bloklarını saklamaktan sorumludur. Her DataNode kendi yerel diskindeki veriden sorumludur. Ayrıca diğer DataNode'lardaki verilerin yedeklerini de barındırır. DataNode'lar küme içerisinde birden fazla olabilir.

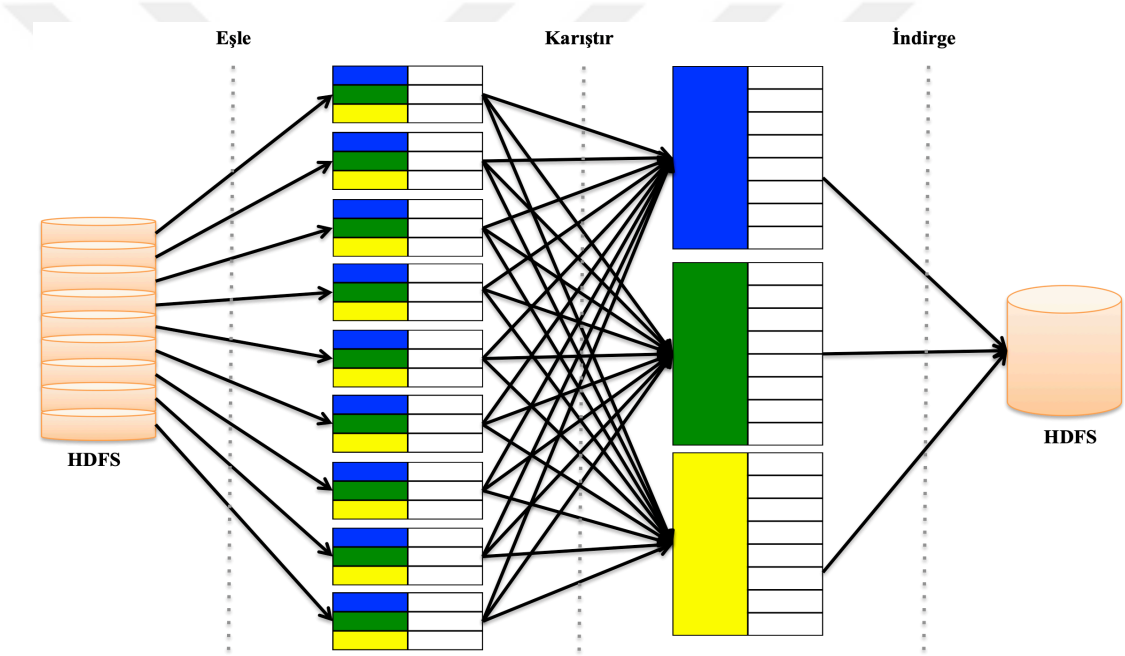


Şekil 1.2. HDFS Mimarisi [10]

1.2.2. Eşle/İndirge programlama modeli

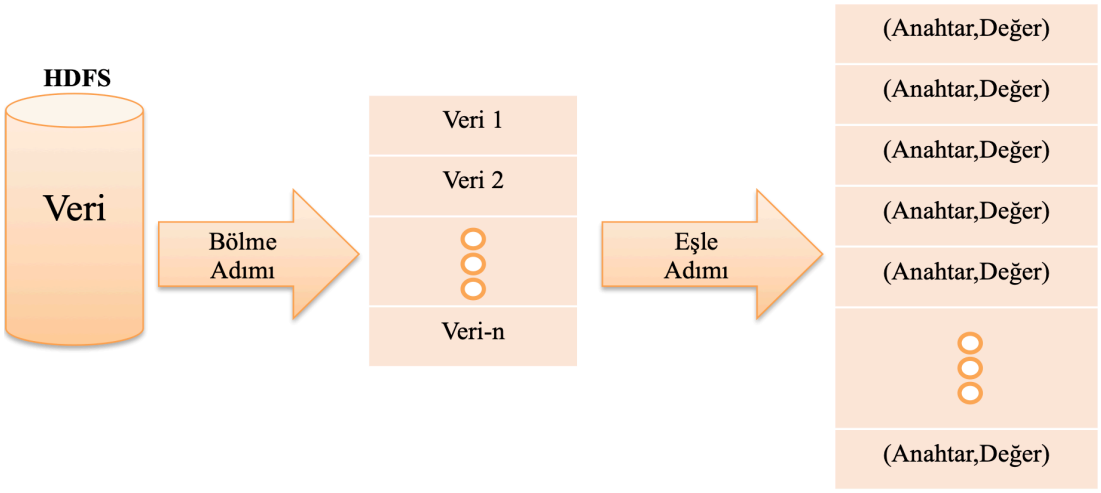
Eşle/İndirge, HDFS üzerindeki büyük verileri işleyebilmek amacıyla kullanılan bir programlama modelidir. Eşle/İndirge programlama modeli mimarisi Şekil 1.3'de gösterildiği gibi; eşle (map), karıştırma (shuffle) ve indirge (reduce) olmak üzere üç adımdan oluşmaktadır.

Eşle fonksiyonu eşle/indirge işinin ilk adımıdır. Eşle fonksiyonu kendi içinde iki (bk. Şekil 1.4) adımdan oluşur; bölme (splitting), girdi verisini alıp küçük veri parçalarına böler ve eşle (mapping), bölünen küçük veriler üzerinde yapılmak istenilen işi paralel olarak yapar [17].

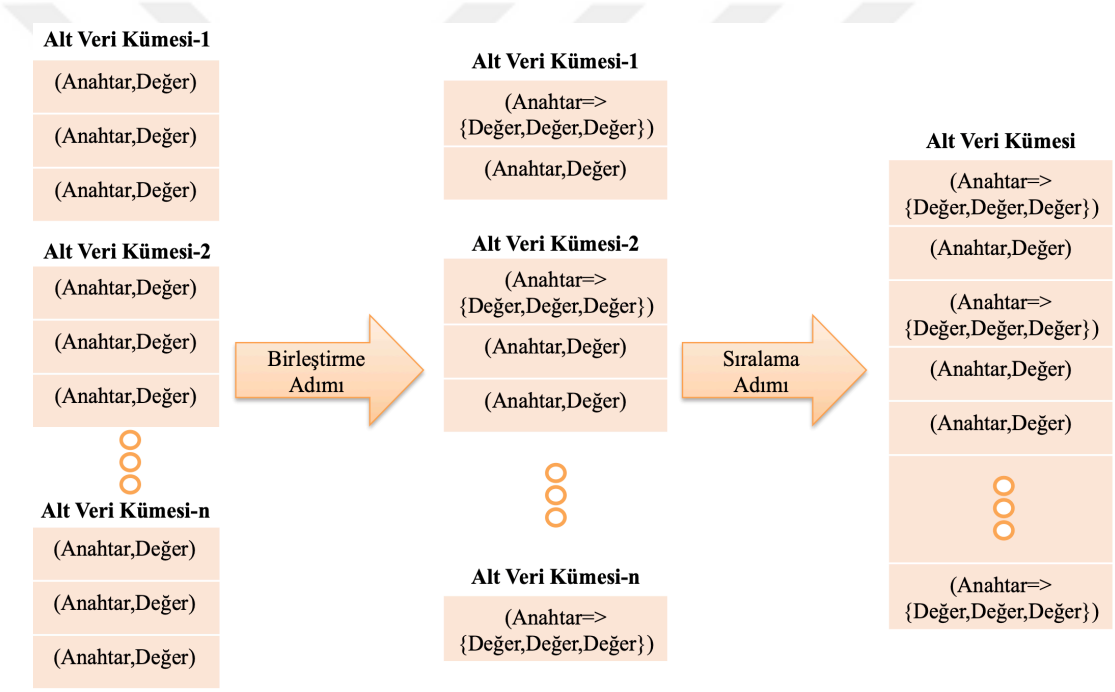


Şekil 1.3. Hadoop Eşle/İndirge Mimarisi

Karıştırma eşle/indirge programlama modelinin ikinci adımıdır, karıştırma fonksiyonu aynı zamanda birleştirme (combine) fonksiyonu olarak da bilinir. Eşle fonksiyonundan çıkan anahtara değer listesini alır ve her anahtar değer için Şekil 1.5'de gösterilen birleştirme (merging) ve sıralama (sorting) adımları uygular. Birleştirme adımında, aynı anahtara sahip bütün değerler bir listeye koyularak birleştirilir. Sıralama adımında, tekil anahtarlar için oluşan listeler kendi içinde sıralanır [17].

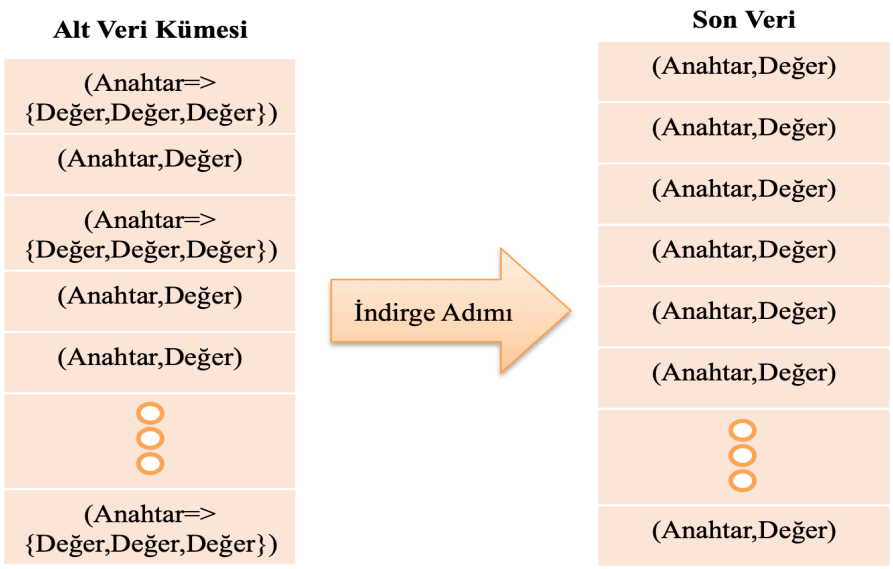


Şekil 1.4. Eşle/İndirge Eşle Adımı [17]



Şekil 1.5. Eşle/İndirge Karıştır Adımı [17]

İndirge, eşle/indirge programlama modelinin son adımıdır. Karıştırma fonksiyonunun üretmiş olduğu sıralı anahtar değer listesini alarak Şekil 1.6'da görüldüğü gibi indirgeme işlemini uygular [17].



Şekil 1.6. Eşle/İndirge İndirge Adımı [17]

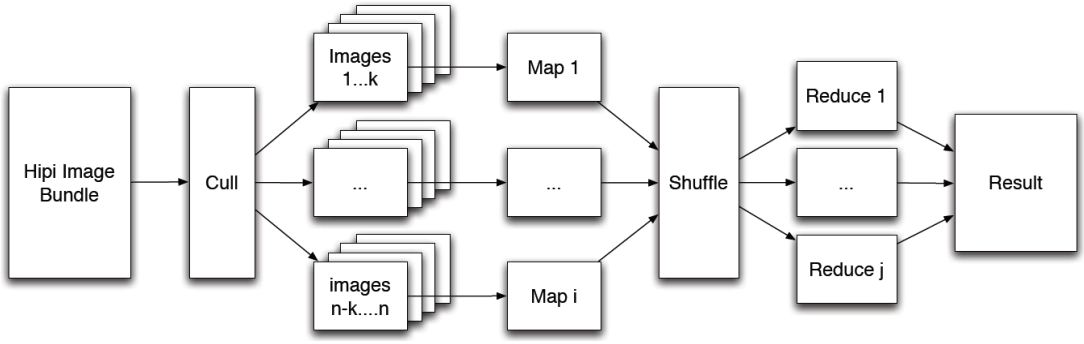
Hadoop, eşle ve indirge'lerden oluşan iş parçacıklarını küme üzerinde dağıtarak aynı anda işlenmesini ve bu işler sonucunda oluşan verilerin tekrar bir araya getirmek için JobTracker ve TaskTracker bileşenlerini kullanır [11-14].

JobTracker, yazılan eşle/indirge programının küme üzerinde dağıtılarak çalıştırılmasından, dağıtılan iş parçacıklarının çalışması sırasında oluşabilecek herhangi bir problemde o iş parçacığının sonlandırılması ya da yeniden başlatılmasından, NameNode'un yardımıyla DataNode'un lokal diskindeki veriye göre en uygun Map işini TaskTracker'a vermekten sorumludur [15].

TaskTracker, DataNode'ların bulunduğu sunucularda çalışır ve JobTracker tarafından verilen iş tamamlamaktan, tamamlanan işin sonuç çıktılarını HDFS üzerinde bir dosya yazmaktan sorumludur [16].

1.3. Hadoop Görüntü İşleme Arayüzü

Hadoop görüntü işleme arayüzü (Hadoop Image Processing Interface-HIPI), Hadoop Eşle/İndirge paralel programlama modeli kullanılarak tasarlanmış bir görüntü işleme kütüphanesidir. HIPI, resimlerin HDFS'te nasıl saklanacağını ve verimli bir şekilde nasıl işleneceğiyle ilgili çözümler sunar. Şekil 1.8'de eşle/indirge HIPI programlama modelinin iş akışı gösterilmiştir [18,19].



Şekil 1.7. Hadoop Eşle/İndirge Hipi Programlama Modeli [18]

Resimler Hipi Image Bundel (HIB) formatına çevrilerek Hipi uygulamasına girdi olarak verilmektedir. Resimleri HIB formatına dönüştürmek için Hipi kütüphaneleri kullanılmaktadır.

Resimler ilk olarak kullanıcının belirlemiş olduğu çözünürlük ve görüntü bilgilerine göre filtrelenir. Bu aşama görüntüler tam olarak çözülmez. Filtreden geçen veriler HibInputFormat sınıfı aracılıyla eşle sınıfına verilir. Eşle sınıfında veriler paralel olarak işlenerek indirge sınıfına gönderilir. İndirge sınıfında veriler tekrardan işlenir ve çıktısı HDFS yazılır.

HIPI ayrıca birçok görüntü işleme algoritması içeren popüler bir açık kaynak kütüphanesi olan OpenCV ile entegrasyon vardır [18].

1.4. Görüntü Örme Yöntemleri

Görüntü örme birden fazla resmin örtüşen kısımları üzerinden birleştirilerek bir resim haline getirilmesi verilen isimdir. Görüntü örme işlemi bilgisayarlarda, askeri ve tıp alanındaki projelerde ve yüksek çözünürlüğe sahip uygu görüntülerinin birleştirilmesi gibi birçok alanda kullanılmaktadır [20]. Günümüzde gelişen teknolojiyle birlikte görüntü örme, panoramik fotoğraf oluşturma özelliği satılan fotoğraf makineleriyle birlikte gelmektedir.

Görüntü örme için ilk algoritmalar Lucas ve Kanade [21] tarafından geliştirilmiştir. Bu algoritmaların farklı türevleri hareket kestirimi algoritmalarında, video özetleme, video sabitleme ve video sıkıştırma dahil olmak üzere çok çeşitli uygulamalarda kullanılmaktadır [22]. Geliştirilen görüntü örme algoritmaları, alan tabanlı (area-

base) ve özellik tabanlı (feature-base) olmak üzere temelde iki ortak yaklaşım altında sınıflandırılmışlardır [23-25].

1.4.1. Alan tabanlı algoritmalar

Alan tabanlı yaklaşımla geliştirilen algoritmalardan birleştirilecek resimlere ait pikseller birebir karşılaştırılır ve karşılaştırma sonucuna göre birleştirilir. Birebir karşılaştırma işlemi algoritmanın maliyetini ve karmaşıklığını artırmaktadır. Bu yöntemle geliştirilen algoritmalar yeteri kadar esnek olmadığından algoritmanın başarısı resim çekilirken oluşan pozlama farklılıklarından büyük ölçüde etkilenir. Aynı zamanda çalışma süreleri çok uzun sürdüğünden gerçek zamanlı uygulamalar için uygun değildirler. Alan tabanlı geliştirilen algoritmanın bir diğer dezavantajda, başlatma, döndürme, vb. işlemler için çok fazla insan etkileşimi vardır [25,26].

1.4.2. Özellik tabanlı algoritmalar

Özellik tabanlı yaklaşımla geliştirilen algoritmalarda resimlere ait tanımlayıcı veriler çıkartılır, birbirleriyle karşılaştırılır ve karşılaştırma sonucuna göre resimler birleştirilir [27,28]. Resimlere ait tanımlayıcı veriler resmin piksellerinin birbirleriyle nasıl ilişkili olduklarına ve geometrik şekillerine göre özellik algılayıcısı tarafından çıkartılır. Geliştirilen algoritmanın başarısı resimlere ait tanımlayıcı verileri çıkartmak için kullanılan yöntemle direk olarak alakalıdır. Resimlere ait tanımlayıcı verileri çıkartmak için kullanılan bir çok algoritma mevcuttur. SURF [29], SIFT, FAST bunlardan en popüler olanlarıdır [30].

2. HADOOP MİMARİSİ KULLANILARAK GELİŞTİRİLEN GÖRÜNTÜ İŞLEME ÇÖZÜMLERİ ÜZERİNE LİTERATÜR TARAMASI

Sweeney C. ve diğerleri çalışmalarında, HIPI çerçevesini iki farklı uygulama ile açıklamışlardır. HIPI'nin amacı büyük ölçekli görüntüleri işlenebilmesini sağlamaktır. Bu görüntü işleme üzerine çalışan araştırmacılara ve öğrencilere büyük bir kolaylık sağlamıştır [31].

Almeer H. yapmış olduğu çalışmada, görüntü işlem algoritmalarını yüksek performanslı bulut bilgi işlem sistemi kullanılarak Hadoop Eşle/İndirge paralel platformda test edilmiş ve kullanılmıştır. Deneyler ve testler, kullanılan çok sayıda büyük görüntünün işlenmesinde ölçeklenebilir ve verimli olduğunu ve aynı zamanda tek PC'li sistemin çalışma süresi ile Hadoop sisteminin çalışma süreleri arasındaki farkı açıkça göstermiştir [32].

Malakar R. ve Vydyanathan N. yapmış oldukları çalışmada, NVDIA tarafından C programlama dilinde GPU çalışacak şekilde geliştirilen Compute Unified Device Architecture (CUDA) modelini Hadoop sistemlerine entegre ederek yüksek performanslı heterojen bir sistem oluşturmaya çalışmışlardır [33].

Demir İ. ve Sayar A., Eşle/İndirge modeli ile görüntü dosyalarının islenebilmesi için geliştirilen yeni bir Hadoop eklentisini (plugin) tanıtmaktadır. Eklenti, görüntü ile ilgili girdi ve çıktı dosya formatları ve girdi dosyalarından kayıtları oluşturan yeni sınıfları içerir. HDFS özellikle az sayıda büyük boyutlu dosyalarla çalışması için tasarlanmıştır. Önerilen teknik, HDFS'de büyük miktarda küçük boyutlu görüntü dosyası kullanımından kaynaklanan performans kayıplarını önlemek için, imgelerin birleştirilerek büyük boyutlu dosyalara dönüştürülmesini temel almıştır. Böylelikle, her bir işleyici çok sayıda imgeyi tek çalışma döngüsünde isleyebilir hale gelir. Önerilen tekniğin etkinliği dağıtık görüntü dosyaları üzerinde yüz saptama (face detection) uygulama senaryosu ile kanıtlanmıştır [34].

Eken S. ve Sayar A., Landsat-7 uydu görüntülerini bir makineden diğerine web servisleri aracılığıyla aktarmak için bir senaryo oluşturmuşlardır. Landsat-7 uydu görüntülerinin hem vektör hem de raster (piksel bazlı) formlarının transferiyle ilgili test ve analizler yapmıştırlar [35].

Eken S. ve Sayar A., raster uydu adası görüntülerini, kenar algılama algoritmalarını kullanarak poligon olarak vektör verisine dönüştüren bir teknik sunmuşturlar. Bu teknik, uydu görüntülerinin veri tabanında vektör nesneleri olarak depolamasını ve kullanılmasını sağlar. Tekniğin etkinliği, gerçek dünyada bir adaya ait uydu görüntüsünde mekânsal-zamana bağlı değişiklikleri saptayan bir sistem üzerine uygulayarak göstermiştirler [36].

Sayar A. ve arkadaşları, uzaktan kaydedilen uydu görüntüleri için özellik tabanlı görüntü birleştirme tekniği önermiştirler. Sistem, çeşitli görüntü işleme tekniklerinin kullanıldığı çoklu adımlardan oluşmaktadır. Speed-up Robust Features (SURF) and Scale Invariant Feature Transform (SIFT), resimlerdeki özelliklerin algılanması ve tanımlanması için alternatif olarak monte edilebilir. Aynı zamanda çoklu spektral ve çoklu zamansal görüntüleri birleştirme konularını ele almışlardır. Önerilen tekniğin etkinliği, Marmara Denizi'nin kısmen örtüşen mozaik uydu görüntülerinin birleştirilmesiyle incelenmiştir. Görüntüler yakın zamanda başlatılan LandSat-8 uydusu tarafından kaydedilmiştir [37].

Eken S. ve Sayar A., LandSat-7 uydusu tarafından kaydedilen görüntülerden bir ada nesnesinin çıkartılması, tanınmasını ve bilgisayar uygulamaları tarafından kolay modellenebilmesi ve işlenmesi sağlayacak çalışmalar yapmıştırlar [38].

Eken S. ve Sayar A., büyük veri mimarisinin sadece metin tabanlı veri madenciliği ve makine öğrenimi algoritmalarıyla geliştirilen uygulamalarında kullanılmadığını, aynı zamanda görüntü işlemede algoritmalarıyla geliştirilen uygulamalarda da kullanıldığını göstermiştirler [39].

Eken S. ve Sayar A., uyduların çekmiş olduğu parça (mozaik) görüntüleri üzerinden, yüksek performansta işlenmesi ve zorlukları ile Eşle/İndirge mimarisine dayalı büyük veri çatısıyla, ölçeklenebilir ve yüksek başarımlı görüntü örme ve nesne çıkarımı mimarilerini incelenmiştir [40].

Eken S. ve arkadaşları, kaynak ve performans problemleri dikkate alınarak çok sayıda raster görüntüyü vektörleştirmek için Eşle/İndirge tabanlı HIPI görüntü işleme dağıtık büyük veri arayüzü kullanılmıştır. Apache Hadoop bu çatının merkezinde yer almaktadır. Böyle bir sistemi gerçeklemek için ilk olarak map fonksiyonu ile girdi ve çıktı formatları tanımlanmıştır. Map fonksiyonları raster görüntüyü vektöre çevirmektedir. Reduce fonksiyonlarına vektörizasyon için ihtiyaç duyulmamıştır. Bant genişliği probleminin negatif etkilerini azaltarak dağıtık hesaplamada daha iyi sonuç almak için raster görüntülerin vektör temsilleri oluşturulmuş ve yatay ölçeklenebilirlik analizleri yapılmıştır [41].

Markonis ve arkadaşları, büyük ölçekli üç boyutlu tıbbi görüntülerin Eşle/İndirge programlama modeliyle işlene bilirliğini ve hızlandığını göstermek için 3 faklı çalışma yapmışlardır; destekçi vektör makinesi (SVM) kullanarak akciğer dokusu sınıflandırma, içerik bazlı tıbbi görüntü indeksleme, doku sınıflandırması için üç boyutlu yönlü dalgacık analizi [42].

Ryu ve arkadaşları, bulut ortamında bulunan videoları paralel bir şekilde işlemek için Hadoop Eşle/İndirge programlama modeli kullanarak yeni bir uygulama çerçevesi oluşturarak video işlemde Hadoop çerçevesinin kullanılabilirliğini ve %75 oranında ölçeklene bilirliğini göstermişlerdir [43].

Yamamoto M. ve Kaneko K. yapmış oldukları çalışmada, video veri tabanlarında bulunan videoların dağıtık bir şekilde işlenmesi ve Hadoop Eşle/İndirge programlama modeliyle geliştirilen çözümler ile ilgili performans değerlendirmeleri yapmışlarıdır [44].

Sozykin A. ve Epanchintsev T. yapmış oldukları çalışmada, görüntü işleme için dağıtık hesaplama yeteneği kullanımını sağlayan eşle/indirge görüntü işleme çerçevesi (mapreduce image processing framework-MIPr) sunmaktadırlar. MIPr, eşle/indirge ve açık kaynaklı Apache Hadoop kütüphanesi kullanılarak geliştirmişlerdir. MIPr, aynı zamanda popüler görüntü işleme algoritmalarını da içermektedir [45].

Rajak R. ve arkadaşları yapmış oldukları çalışmada uzaktan algılanan uydu görüntülerinin Hadoop ortamlarında saklanması, işlenmesi ve işlenen görüntü

çıktılarının hbase saklanması için bir Eşle/İndirge ortamı sunmuşlarıdır. Hadoop sayesinden sıradan sunucu kümeleri üzerinde çalışmalar yapılarak hız ve performans kazanımları elde edilmiştir [46].

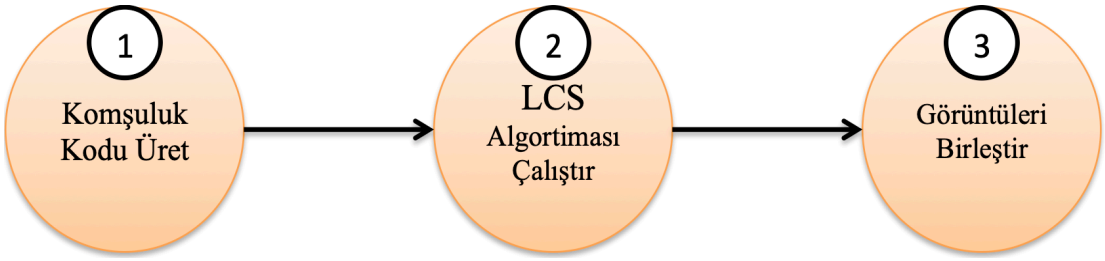
Banaei S. ve Moghaddan H. yapmış oldukları çalışmada, açık kaynaklı dağıtılmış bir hesaplama çerçevesi olarak sonsuz miktarda hesaplama düğüm kümesiyle büyük miktarda görüntünün işlenmesine olanak sağlayan Hadoop çerçevesini, mevcut çalışmaları, avantajlarını ve dezavantajlarını incelemişlerdir [47].

Kocakulak H. ve Taşkaya T. yapmış oldukları çalışmada, büyük miktarda verilerle yoğun hesaplamalar gerektiren balistik görüntü karşılaştırma işlemi için Apache Hadoop çerçevesi altında bir Eşle/İndirge çözümü geliştirmişlerdir. Geliştirmiş oldukları çözümünle önemli derecede kaynak kazanımı ve performans artışı sağlamışlardır [48].

3. HADOOP EŞLE/İNDIRGE PROGRAMLAMA MODELİ KULLANILARAK GELİŞTİRİLEN GÖRÜNTÜ ÖRME ÇÖZÜMÜ

Tez süresince birden fazla görüntünün birleştirilmesine yönelik çalışmalar gerçekleştirilmiştir. Öncelikle örme işlemi JVM'de tek thread çalışacak şekilde geliştirilmiş daha sonra JVM'de birden fazla threadle çalışacak şekilde paralelize edilmiş ve sonrasında Hadoop ortamlarında Eşle/İndirge programlama modeliyle geliştirme yapılmıştır.

Geliştirilen çözümlerde Şekil 3.1'de gösterildiği gibi görüntü örme işlemi temelde 3 adımdan oluşmaktadır. Birinci adımda resimlere ait pikselleri metin olarak ifade edebilmek için piksellerin komşu pikselleriyle benzerlikleri göz ününde bulundurularak her bir piksel için bir komşuluk kodu üretilmiştir. İkinci adımda resimlere ait pikseller için üretilen komşuluk kodları En Uzun Ortak Alt Dizi (Longest Common Substring) algoritmasıyla karşılaştırılmıştır. Üçüncü adımdaysa LCS algoritması sonucunda bulunan referans değerlere göre resimler birleştirilmiştir.



Şekil 3.1. Görüntü Örme Adımları

Bu kısımda öncelikli olarak resimlere ait piksellerden nasıl komşuluk kodu üretildiğine, En Uzun Ortak Alt Dizi ve En Uzun Ortak Küme algoritmalarının nasıl çalıştığından sonrasında geliştirilen 3 farklı çözümden bahsedilecektir.

3.1. Piksellerin Komşuluk Kodlarının Üretilmesi

Resimlere ait piksellerin komşuluk kodu, pikselin komşu pikselleriyle olan benzerliğine göre iteratif (Şekil 3.3 Algoritma 1 bk.) bir şekilde üretilen koddur. Piksellerin komşuluk kodlarının üretilebilmesi için resimler kenar belirleme (Edge

Detection) yöntemiyle tekrardan oluşturulur. Yeni oluşan resimde kenarlar kırılımlarını belirten pikseller beyaz diğer pikseller siyahtır. Beyaz pikselin RGB değeri '255', siyah pikselin RGB değeri '0' dır. Değeri '255' olan bir pikselin komşuları saat yönünde kontrol edilir ve değeri '255' olan komşusunun bulunduğu yöne göre Tablo 3.1 de belirtilen kod pikselin komşuluk koduna eklenir ve aynı kontrol işlemleri bulunan komşular için iteratif bir şekilde tekrarlanır.

Tablo 3.1. Yönlere Göre
Komşuluk Kod Değerleri

5	6	7
4	255	0
3	2	1

Değeri '255' olan bir piksel için komşuluk kodu üretme işlemi adım adım incelenecek olursa;

Y\X	0	1	2	3
0	0	0	0	0
1	0	255	255	0
2	0	0	255	0
3	0	0	255	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0

(a)

Y\X	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	255	255	0
6	0	0	255	0
7	0	0	255	0

(b)

Şekil 3.2. Komşuluk Kod Üretim Adımları a) Görüntü 1 b) Görüntü 2

Algoritma 1. Iteratif Komşuluk Kodu Üretme Algoritması

```
 1: function KomsulukKoduUret(resim,x,y)
 2:   kod ← ""
 3:   zero ← 0
 4:   while loop(true)
 5:     resim[x,y] ← zero
 6:     if resim[x+1,y]==255 then
 7:       kod ← kod + "0"
 8:       x ← x+1
 9:       continue
10:     end if
11:     if resim[x+1,y+1]==255 then
12:       kod ← kod + "1"
13:       x ← x+1
14:       y ← y+1
15:       continue
16:     end if
17:     if resim[x,y+1]==255 then
18:       kod ← kod + "2"
19:       y ← y+1
20:       continue
21:     end if
22:     if resim[x-1,y+1]==255 then
23:       kod ← kod + "3"
24:       x ← x-1
25:       y ← y+1
26:       continue
27:     end if
28:     if resim[x-1,y]==255 then
29:       kod ← kod + "4"
30:       x ← x-1
31:       continue
32:     end if
33:     if resim[x-1,y-1]==255 then
34:       kod ← kod + "5"
35:       x ← x-1
36:       y ← y-1
37:       continue
38:     end if
39:     if resim[x,y-1]==255 then
40:       kod ← kod + "6"
41:       y ← y-1
42:       continue
43:     end if
44:     if resim[x+1,y-1]==255 then
45:       kod ← kod + "7"
46:       x ← x+1
47:       y ← y-1
48:       continue
49:     end if
50:     break
51:   end loop
52:   return kod
53: end function
```

Şekil 3.3. Iteratif Komşuluk Kodu Üretme Algoritması

Şekil 3.2'de gösterilmekte olan Görüntü 1'in [1,1] numaralı indeksteki pikselin değeri '255' tir, bu pikselin komşuları kontrol edilir, [2,1] indeksinde değeri '255' olan komşusu için [1,1] indeksteki pikselin komşuluk koduna '0' eklenir, [2,1] indeksinin komşuları kontrol edilir, [2,2] indeksinde değeri '255' olan komşusu için

[1,1] indeksteki pikselin komşuluk koduna ‘2’ eklenir[2], [2,2] indeksinin komşuları kontrol edilir, [2,3] indeksinde değeri ‘255’ olan komşusu için [1,1] indeksteki pikselin komşuluk koduna ‘2’ eklenir, [2,3] indeksinin değeri ‘255’ komşusu olmadığı için [1,1] indeksli piksel için komşuluk kod üretme işlemi bitmiş olur. Kontrol edilen ‘255’ değerindeki pikseller tekrardan kontrol edilmemek için değeri ‘0’ olan siyah ile değiştirilir. [1,1] indeksinde bulunan piksel için üretilen komşuluk kodu ‘022’ dir.

3.2. En Uzun Ortak Alt Dizi Algoritması

En uzun ortak alt dizi algoritması(Longost Common Substring-LCS), verilen iki dizi arasında her iki dizide bulunan en uzun ortak alt diziyi bulan algoritmadır (Şekil 3.4. Algoritma 2’ye bk.). Bulunan alt dizilerin elemanlarının ardı ardına olma zorunluluğu vardır. LCS algoritması O(nm) karmaşıklığa sahiptir ve O(nm) depolama alanı kullanır [49-50].

“wertyabcdr” ve “wklrabcderf” dizileri için LCS değeri 4 elemanlı “abcd” dizisidir.

Algoritma 2. Longest Common Substring Algoritması

```
 1: function LCS(inputStr1,inputStr2)
 2:   m ← inputStr1.length
 3:   n ← inputStr2.length
 4:   LCStuff ← dizi(0..m+1,0..n+1)
 5:   result ← 0
 6:   for i=0 to m do
 7:     for j=0 to n do
 8:       if i==0 or j==0 then
 9:         LCStuff[i,j] ← 0
10:       else if inputStr1.substr(i-1,i) == inputStr2.substr(j-1,j) then
11:          LCStuff[i,j] ← LCStuff[i-1,j-1] + 1
12:         result ← MAX(result,LCStuff[i,j])
13:       else
14:         LCStuff[i,j] ← 0
15:       end if
16:     end for
17:   end for
18:   return result
19: end function
```

Şekil 3.4. Longest Common Substring Algoritması

3.3. En Uzun Ortak Küme Algoritması

En uzun ortak Küme algoritması(Longest Common Subsequence), verilen iki dizi arasında her iki dizide bulunan en uzun ortak kümeyi bulan algoritmadır (Şekil 3.5

Algoritma 3'e bk.). Bulunan küme elemanlarının ardı ardına olma zorunluluğu yoktur. Longest Common Subsequence algoritması O(nm) karmaşıklığa sahiptir ve O(nm) depolama alanı kullanır [51-52].

"wertyabcdr" ve "wklrabcderf" dizileri için Longest Common Subsequence değeri 7 elemanlı "wrabcdr" dizisidir.

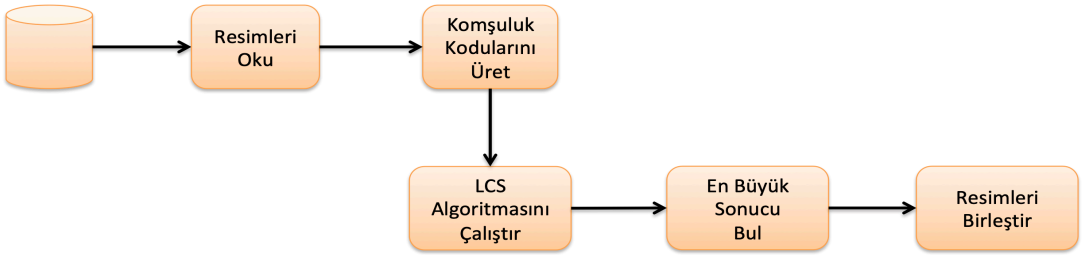
Algoritma 3. Longest Common Subsequnce Algoritması

```
 1: function LCS(inputStr1,inputStr2)
 2:   m ← inputStr1.length
 3:   n ← inputStr2.length
 4:   LCStuff ← dizi(0..m+1,0..n+1)
 5:   result ← 0
 6:   for i=0 to m do
 7:     for j=0 to n do
 8:       if i==0 or j==0 then
 9:         LCStuff[i,j] ← 0
10:       else if inputStr1[i-1] == inputStr2[j-1] then
11:          LCStuff[i,j] ← LCStuff[i-1,j-1] + 1
12:       else
13:         LCStuff[i,j] ← MAX(LCStuff[i-1,j],LCStuff[i,j-1])
14:       end if
15:     end for
16:   end for
17:   result ← LCStuff[m,n]
18:   return result
19: end function
```

Şekil 3.5. Longest Common Subsequence Algoritması

3.4. JVM Tek Thread Olarak Geliştirilen Çözüm

Tez sürecinde ilk olarak Java programla dilinde resimlerin metin olarak ifade edilmesi ve resimlerden çıkartılan metinlerin karşılaştırılması üzerine geliştirmeler yapıldı. Bu süreçte komşuluk kodu üretme algoritmasıyla resimlerin pikselleri kullanılarak resimler metin olarak ifade edildi. Komşuluk kodu algoritması geliştirilirken rekursif ve iteratif programlama yöntemleri denenerek test edildi. Üretilen komşuluk kodlarını karşılaştırmak için Longest Common Subsequnece ve Longest Common Substring algoritmaları denenerek test edildi. Uygulamanın bütün olarak çalışması göz önünde bulundurulduğunda resimlere ait komşuluk kodu üretmek için en efektif yöntemin iteratif programlama olduğuna ve üretilen komşuluk kodlarını karşılaştırmak için ise Longest Common Substring algoritması olduğuna karar verildi.



Şekil 3.6. Tek Thread Olarak Geliştirilen Çözüm Akışı

JVM'de tek thread olarak Java programla dilinde geliştirilen çözüm akış şeması Şekil 3.6'da gösterilmiştir. Görüntüler ilk olarak dosya sisteminden BufferedImage olarak okunuyor, dosya sisteminden okunan görüntülerden komşuluk kodu üretebilmek için görüntüler Şekil 3.7'de gösterildiği gibi edge detection yöntemiyle siyah beyaz formata dönüştürülüyor.

(a) (b)

Şekil 3.7. Siyah Beyaz Görüntü Dönüşümü a) Normal Görüntü b) Siyah Beyaz Görüntü

Bölüm 3.1'de anlatıldığı gibi siyah beyaz görüntünün pikselleri kullanılarak komşuluk kodları üretiliyor. Görüntüler için üretilen komşuluk kodları Map veri türüne Tablo 3.2'de gösterilen formatta ekleniyor. Resimler için üretilen komşuluk kodlarının listelere eklerken komşuluk kodunun uzunluğuna bağlı bir filtreleme işlemi yaparak LCS algoritmasının çalışma süresini ve kartezyen çarpım işleminin maliyetini azaltılmıştır.

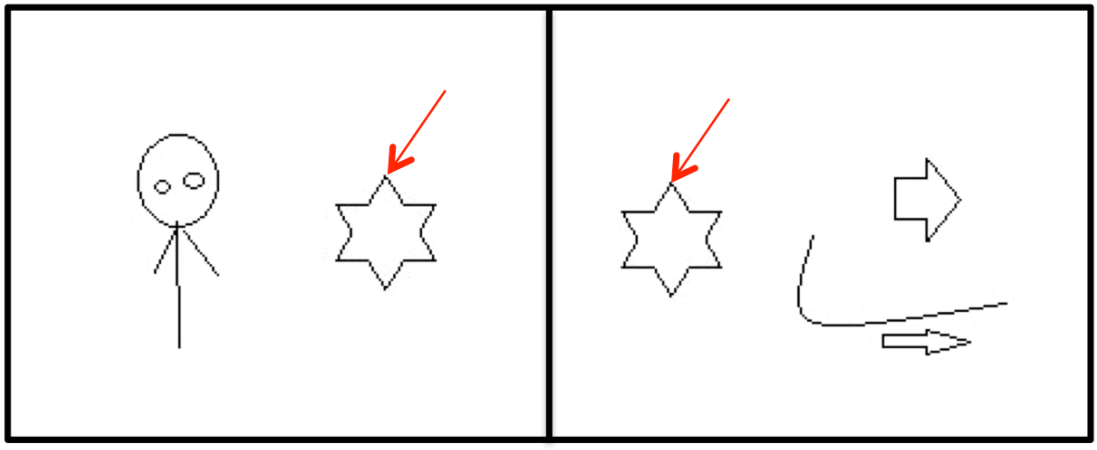
Farklı görüntüler için üretilen komşuluk kodları kartezyen çarpıma sokularak komşuluk kodu çiftleri oluşturuluyor. Oluşturulan çifter LCS algoritmasıyla karşılaştırılıyor. En büyük değere sahip karşılaştırma sonuç resimlerin muhtemel

birleştirme noktası olarak kabul ediliyor ve görüntüler bu piksellerin koordinatları kullanılarak birleştiriliyor.

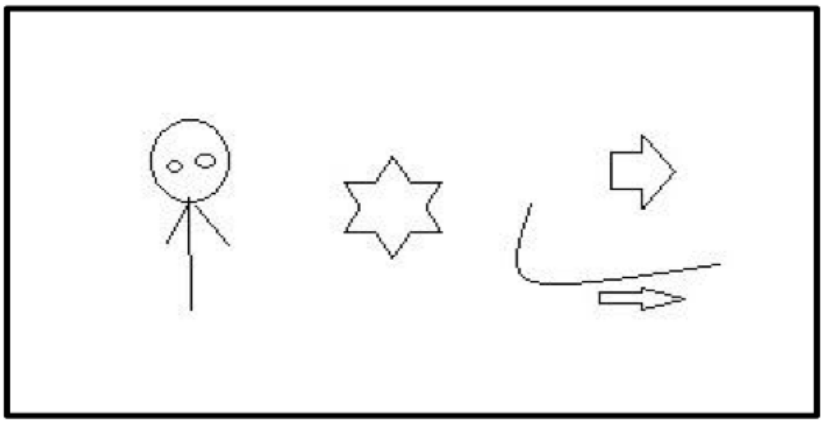
Tablo 3.2. Resimler İçin Üretilmiş Komşuluk Listesi

Key Görüntülerin ismi:	Value: Resim ait pikseller için üretilen komşuluk kod bilgilerini içeren liste.(List<Pixel>, Pixel={X Kordinatı,Y Kordinatı, Komşuluk Kodu})
Resim 1	$\{\{x_1,y_1,KK\},\{x_2,y_2,KK\},\{x_3,y_3,KK\},\{x_4,y_4,KK\},..,\{x_n,y_n,KK\}\}$
Resim 2	$\{\{x_1,y_1,KK\},\{x_2,y_2,KK\},\{x_3,y_3,KK\},\{x_4,y_4,KK\},..,\{x_m,y_m,KK\}\}$

Şekil 3.8'de görüntü 1 ve görüntü 2 birleştirmek istenmiştir. Görüntüler için komşuluk kodları üretilmiş ve LCS algoritmasıyla karşılaştırılıyor. En uzun ortak alt diziye sahip komşuluk kodları kırmızı oklar ile gösterilen pikseller kullanılarak üretilmiştir. Görüntüler bu piksellerin koordinatları kullanarak Şekil 3.9'da gösterildiği şekilde birleştirilmiştir.



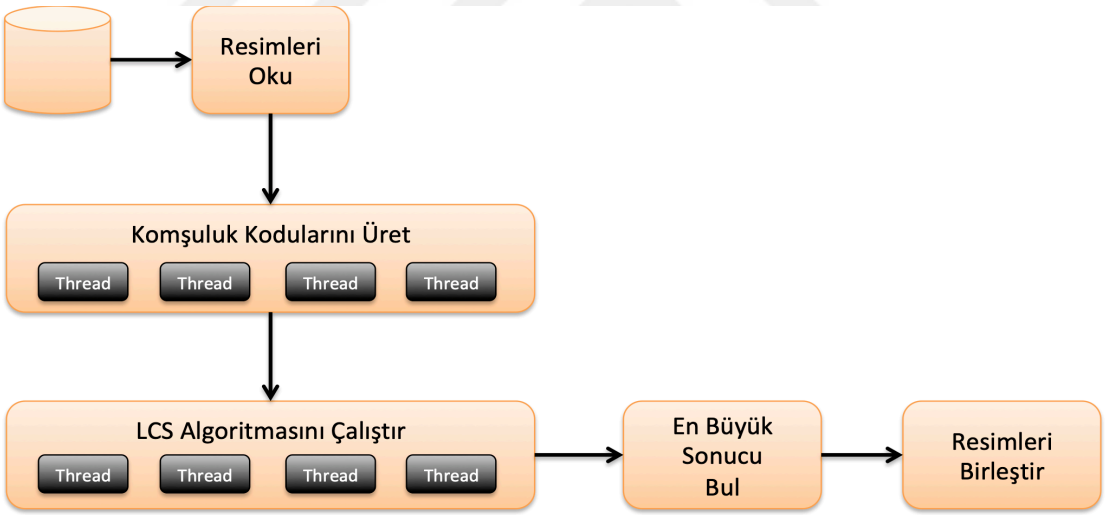
Şekil 3.8. Birleştirmek İstenen Örnek Görüntüler a) Görüntü 1 b) Görüntü 2



Şekil 3.9. Birleştirilmiş Yeni Görüntü

3.5. JVM Çoklu Thread Olarak Geliştirilen Çözüm

Görüntülerin birleştirilmesi için kullanılan LCS algoritmasıyla geliştirilmiş çözümün adımları incelenmiştir sistemin daha hızlı çalışması için bazı adımları Şekil 3.10'daki gibi paralelleştirilmiştir. Paralelleştirme için threadlerden faydalanılmıştır.



Şekil 3.10. Çoklu Thread Olarak Geliştirilen Çözüm Akışı

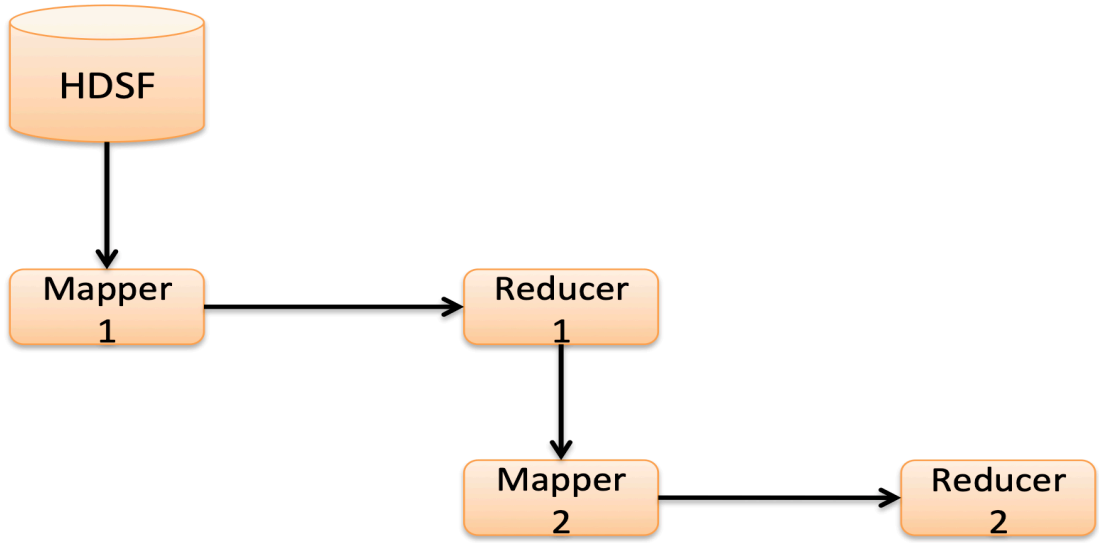
Çoklu thread olarak geliştirilen çözümde tekli thread olarak geliştirilen çözümden farklı olarak görüntülerden komşuluk kod üretilmesi threadler kullanılarak paralelleştirmiştir ve komşuluk kodlarının kartezyen çarpıma sokularak oluşturulan komşuluk kod çiftlerinin LCS algoritmasıyla karşılaştırılması yine threadler kullanılarak paralelleştirilmiştir.

JVM çoklu thread olarak geliştirilen çözümün amacı tek thread olarak geliştirilen çözümle performans karşılaştırması yapmak ve aynı zamanda LCS algoritması

kullanılarak önerilen görüntü birleştirme çözümünün Eşle/İndirge programla modelenine uygulanabilirliğiyle ilgili ön çalışma yapmak.

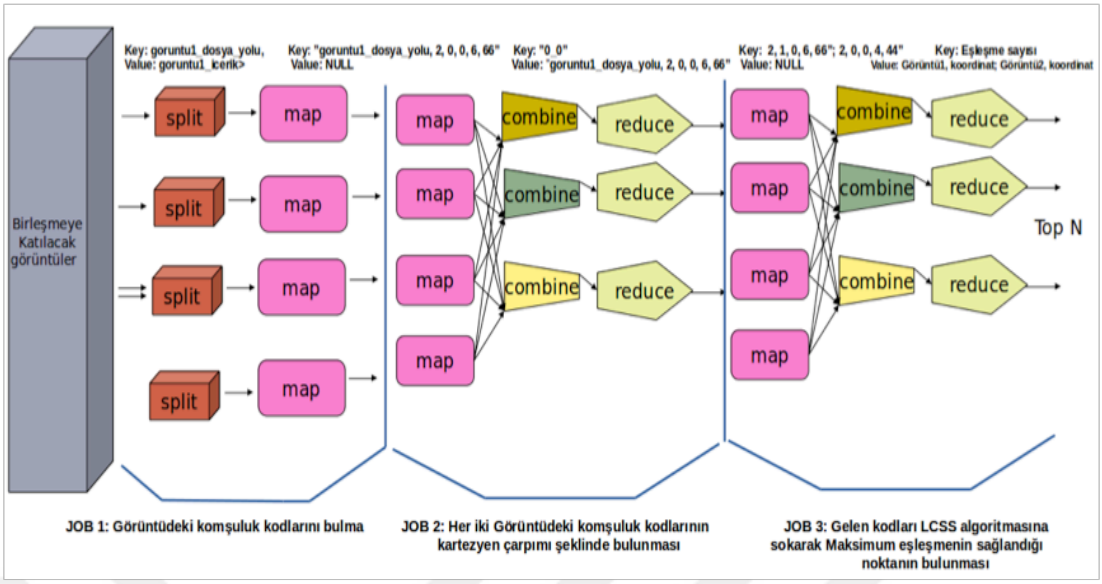
3.6. Hadoop Eşle/İndirge Programlama Modeliyle Geliştirilen Çözüm

Gerçek dünya problemlerinde sadece tek bir Eşle/İndirge oluşan bir iş (job) tanımlamak yeterli değildir. Birçok problemde şu şekilde bir yapı gerekmektedir: Map1→Reduce1→Map2→Reduce2→Map3→Reduce3→… Her bir iş için farklı driver'lar yazmak gerekir. Bu şekilde bir yaklaşım literatürde zincirleme Eşle/İndirge (chaining Eşle/İndirge) olarak geçmektedir. Bir iş bittikten sonra çıktıyı path'a yazar, akabindeki diğer iş ilkinin çıktısını giriş olarak alır. Bütün işler birbirini takip ederek çalışır ve genel problem çözülmüş olur. Şekil 3.11'de arka arkaya birer tane eşle ve indirgeden oluşan iki işe ait akış verilmiştir.



Şekil 3.11. Zincirleme Eşle/İndirge Akışı

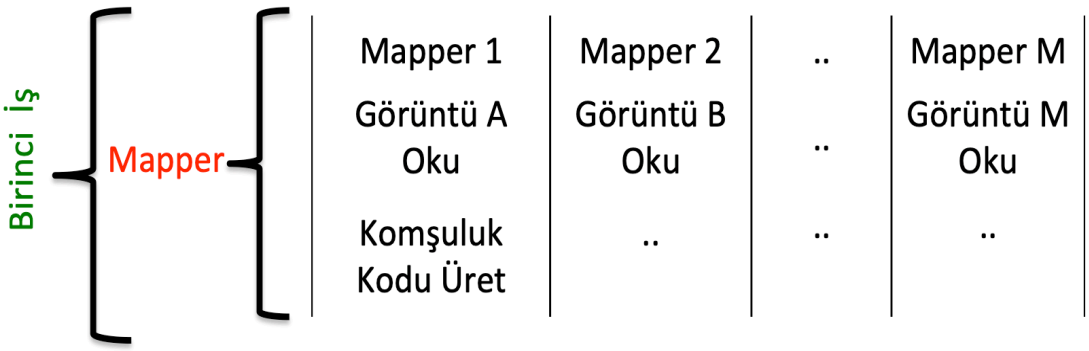
Geliştirilen çözüm Şekil 3.12'de görüldüğü gibi 3 işten oluşmaktadır. Her iş bir önceki işin üretmiş olduğu çıktıyı girdi olarak kullanmaktadır.



Şekil 3.12. Zincirleme Eşle/İndirge Akışı

3.6.1. Birinci eşle/indirge işi

İlk iş, tek eşle metodu içermektedir (Şekil 3.13 bk.). Eşle metodu girdi key-value çifti, kayıt okuyucu sınıfının döndürmüş olduğu görüntülere ait dosya yolu ve görüntü içeriği çiftidir. Görüntü içindeki piksel değerlerine göre komşuluk kodları çıktısı üretilir. Çıktıdaki key değeri; görüntünün dosya yolu, görüntüdeki toplam komşuluk kodları sayısı, indeks değeri (komşuluk kodu kadar 0'dan başlayarak artan), görüntüdeki ilgili komşuluk kodu için referans alınan pikselin başlangıç ve bitiş koordinatları ile komşuluk kodunun kendinden oluşmaktadır. Çıktıdaki value değeri ise NULL'dır.



Şekil 3.13. Birinci İş

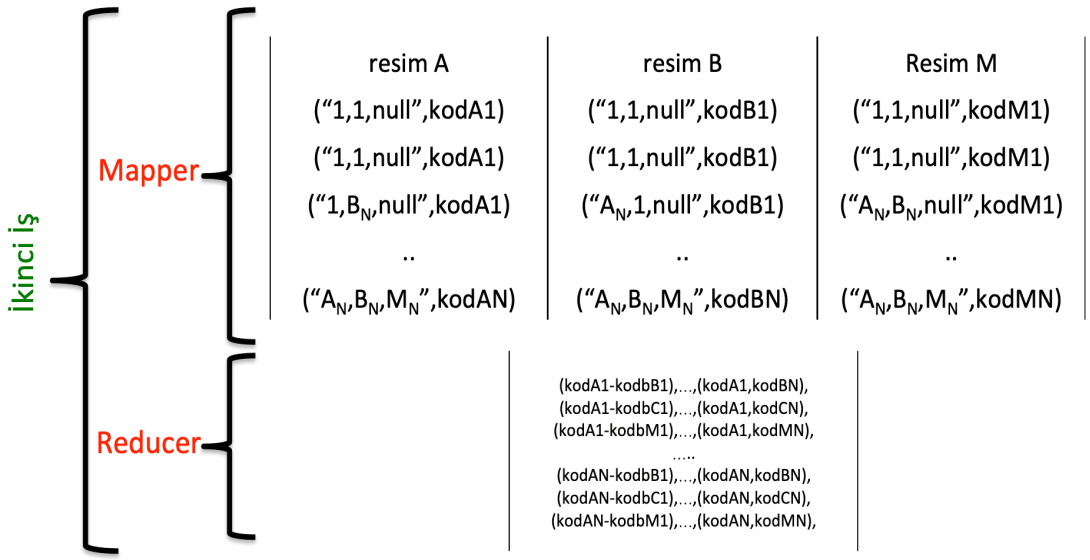
Her görüntü farklı bir eşle tarafından okunup işlenerek paralel bir şekilde resimlerinde işlenmesi sağlanmıştır.

Tablo 3.3. Birinci İş Eşle Girdi/ Çıktı Çiftleri Formatı

Eşle girdi key-value çifti	Eşle çıktı key-value çifti
<görüntü1_dosya_yolu, görüntü 1_icerik> < görüntü2_dosya_yolu, görüntü 2_icerik>	<"görüntü1_dosya_yolu, 2, 0, 0, 6, 66", NULL> <"görüntü1_dosya_yolu, 2, 1, 1, 0, 42", NULL> <"görüntü2_dosya_yolu, 2, 0, 0, 4, 44", NULL> <"görüntü2_dosya_yolu, 2, 1, 2, 0, 42", NULL>

3.6.2. İkinci eşle/indirge işi

İkinci iş eşle ve indirge metodundan oluşmaktadır (Şekil 3.14 bk.).. Bu işte, görüntülerden elde edilen komşuluk kodlarının birbirleriyle kartezyen çarpımı yapılarak LCS algoritmasında çalışacak çiftler oluşturulmuştur. Eşle metodunun girdi key-value çifti bir önceki işteki eşle metodunun çıktı key-value çiftine eşittir. Eşle metodu çıktı key'i görüntüdeki kodların kartezyen şekilde bir araya gelmesini ifade eden bir string, value ise görüntülerdeki komşuluk kodunu içermektedir. İndirge aşamasında aynı key değerine sahip her iki görüntüdeki komşuluk kodlarının bir araya getirilmesi sağlanmaktadır. Tablo 3.4 ve Tablo 3.5'de ikinci işe ait Eşle/İndirge girdi çıktı key-value değerleri gösterilmiştir.



Şekil 3.14. İkinici İş

Tablo 3.4. İkinci İş Eşle Girdi ve Çıktı Key-Value Değerleri

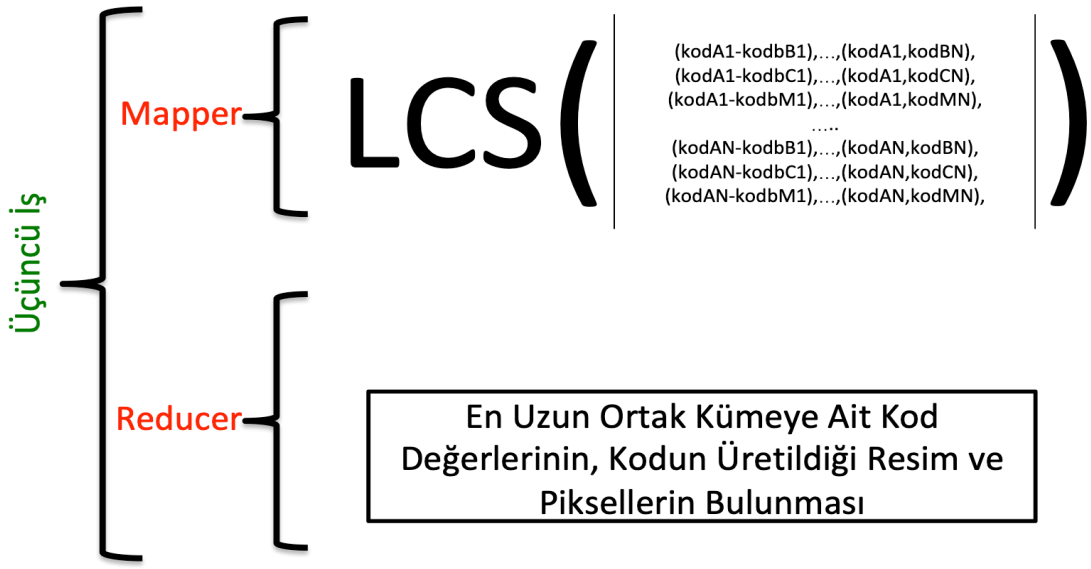
Eşle girdi key-value çifti	Eşle çıktı key-value çifti
<”goruntu1_dosya_yolu, 2, 0, 0, 6, 66”, NULL> <”goruntu1_dosya_yolu, 2, 1, 1, 0, 42”, NULL> <”goruntu2_dosya_yolu, 2, 0, 0, 4, 44”, NULL> <”goruntu2_dosya_yolu, 2, 1, 2, 0, 42”, NULL>	<”0_0”, ”goruntu1_dosya_yolu, 2, 0, 0, 6, 66”> <”0_0”, ”goruntu2_dosya_yolu, 2, 0, 0, 4, 44”> <”0_1”, ”goruntu1_dosya_yolu, 2, 0, 0, 6, 66”> <”1_0”, ”goruntu1_dosya_yolu, 2, 1, 1, 0, 42”> <”1_0”, ”goruntu2_dosya_yolu, 2, 0, 0, 4, 44”> <”1_1”, ”goruntu1_dosya_yolu, 2, 1, 1, 0, 42”> <”1_1”, ”goruntu2_dosya_yolu, 2, 1, 2, 0, 42”>

Tablo 3.5. İkinci İş İndirge Girdi Çıktı Key-Value Değerleri

İndirge girdi key-value çifti	İndirge çıktı key-value çifti
<”0_0”, ”goruntu1_dosya_yolu, 2, 1, 0, 6, 66”> <”0_0”, ”goruntu2_dosya_yolu, 2, 0, 0, 4, 44”> <”0_1”, ”goruntu1_dosya_yolu, 2, 1, 0, 6, 66”> <”0_1”, “goruntu2_dosya_yolu, 2, 1, 2, 0, 42”> <”1_0”, ”goruntu1_dosya_yolu, 2, 1, 1, 0, 42”> <”1_0”, ”goruntu2_dosya_yolu, 2, 0, 0, 4, 44”> <”1_1”, ”goruntu1_dosya_yolu, 2, 1, 1, 0, 42”> <”1_1”, ”goruntu2_dosya_yolu, 2, 1, 2, 0, 42”>	<”goruntu1_dosya_yolu, 2, 1, 0, 6, 66”; ”goruntu2_dosya_yolu, 2, 0, 0, 4, 44”, NULL> <”goruntu1_dosya_yolu, 2, 1, 0, 6, 66”; “goruntu2_dosya_yolu, 2, 1, 2, 0, 42”, NULL> <”goruntu1_dosya_yolu, 2, 1, 1, 0, 42”; ”goruntu2_dosya_yolu, 2, 0, 0, 4, 44”, NULL> <”goruntu1_dosya_yolu, 2, 1, 1, 0, 42”; ”goruntu2_dosya_yolu, 2, 1, 2, 0, 42”, NULL>

3.6.3. Üçüncü eşle/indirge işi

Üçüncü iş eşle ve indirge metotlarından oluşmaktadır (Şekil 3.15 bk.).. Eşle metodunda ilgili komşuluk kodları üzerinde LCS algoritmasıyla karşılaştırılarak aralarındaki benzerlik bulunmaktadır. Çıktı key değeri olarak bu sayı; value olarak ise karşılaştırılan komşuluk kodlarının üretildiği resmin dosya yolu, algoritmaya giren komşuluk kodu için referans alınan pikselin başlangıç ve bitiş koordinatları aktarılır. İndirge aşamasında ise maksimum eşleşme sağlayan 5 komşuluk kodu ve iki görüntünün hangi koordinatlardan birleştirileceği bilgisi elde edilir. Tablo 3.6’da üçüncü işe ait eşle girdi ve çıktı key-value değerlerini göstermektedir.



Şekil 3.15. Üçüncü İş

Tablo 3.6. Üçüncü İş Eşle Girdi Çıktı Key-Value Değerleri

Eşle girdi key-value çifti	**Eşle girdi key-value çifti**
<”goruntu1_dosya_yolu, 2, 1, 0, 6, 66”; ”goruntu2_dosya_yolu, 2, 0, 0, 4, 44”, NULL> <”goruntu1_dosya_yolu, 2, 1, 0, 6, 66”; “goruntu2_dosya_yolu, 2, 1, 2, 0, 42”, NULL> <”goruntu1_dosya_yolu, 2, 1, 1, 0, 42”; ”goruntu2_dosya_yolu, 2, 0, 0, 4, 44”, NULL> <”goruntu1_dosya_yolu, 2, 1, 1, 0, 42”; ”goruntu2_dosya_yolu, 2, 1, 2, 0, 42”, NULL>	<0; “goruntu1_dosya_yolu”, 0, 6; goruntu2_dosya_yolu, 1, 0”> <0; “goruntu1_dosya_yolu”, 0, 6; goruntu2_dosya_yolu, 2, 0”> <1; “goruntu1_dosya_yolu”, 1, 0; goruntu2_dosya_yolu, 1, 0”> <2; “goruntu1_dosya_yolu”, 1, 0; goruntu2_dosya_yolu, 2, 0”>

4. TESTLER VE PERFORMANS ÇIKTILARI

Geliştirilen Görüntü Örme uygulamalarıyla farklı boyutta resimler kullanılarak testler yapılmıştır. JVM'de çalışan uygulamalar için Intel Core i7 2.8 GHz işlemciye, 16GB 1600 MHz DDR3 belleğe sahip MacBook Pro bilgisayarı kullanılmıştır. Hadoop ortamı için geliştirilen uygulama, Apache Hadoop'un Cloudera sürümünün kurulu oldu 3 node'lu sanal sunucu küme üzerinde çalıştırılmıştır. Hadoop kümesinde bulunan 3 sanal sunucunun özellikleri Tablo 4.1'deki gibidir.

Tablo 4.1. Hadoop Kümesi Sunucu Özellikleri

Hostname	OS Versiyon	Hadoop Versiyon	CPU	RAM	DISK
hadoop1	Centos 7	Hadoop 3.0.0 CDH 6.0.1	Intel(R) Xeon(R) CPU X7550 @ 2.00GHz	32 GB	92 GB
hadoop2	Centos 7	Hadoop 3.0.0 CDH 6.0.1	Intel(R) Xeon(R) CPU X7550 @ 2.00GHz	16 GB	92 GB
hadoop3	Centos 7	Hadoop 3.0.0 CDH 6.0.1	Intel(R) Xeon(R) CPU X7550 @ 2.00GHz	16 GB	92 GB

Testlerde uygulamaların başarılı bir şekilde birleştirdiği farklı boyutlara sahip görüntüler (Şekil 4.1 ve Şekil 4.2 bk.) kullanılmıştır. Farklı boyuttaki görüntüler Şekil 3.1'de belirtilen görüntü örme adımlarına göre kendi içlerinde karşılaştırılmıştır. Çoklu thread'de thread sayısı 3 olarak ayarlanarak testler yapılmıştır.

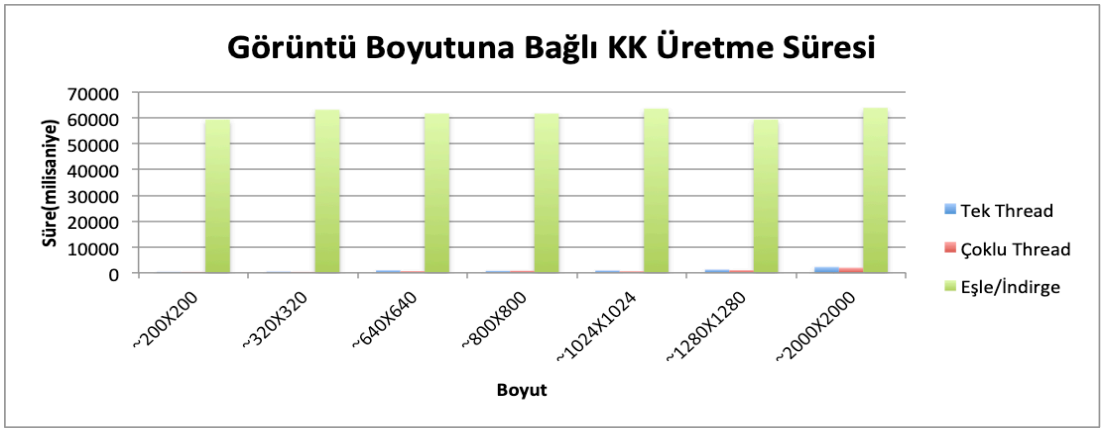
Şekil 4.1. Birleştirilmek İstenen Örnek Görüntüler

Şekil 4.2. Örnek Görüntülerin Birleştirilmiş Hali

Üç farklı uygulama görüntü boyutuna bağı komşuluk kodu üretme süreleri Tablo 4.2 ve Şekil 4.1'deki gibidir.

Tablo 4.2. Görüntü Boyutuna Bağlı KK Üretme Süresi

	Tek Thread (milisaniye)	Çoklu Thread (3 Thread, milisaniye)	Eşle/İndirge (milisaniye)
~200X200	401	351	59382
~320X320	482	293	63149
~640X640	964	636	61696
~800X800	762	795	61684
~1024X1024	890	600	63601
~1280X1280	1246	994	59360
~2000X2000	2246	1950	63889

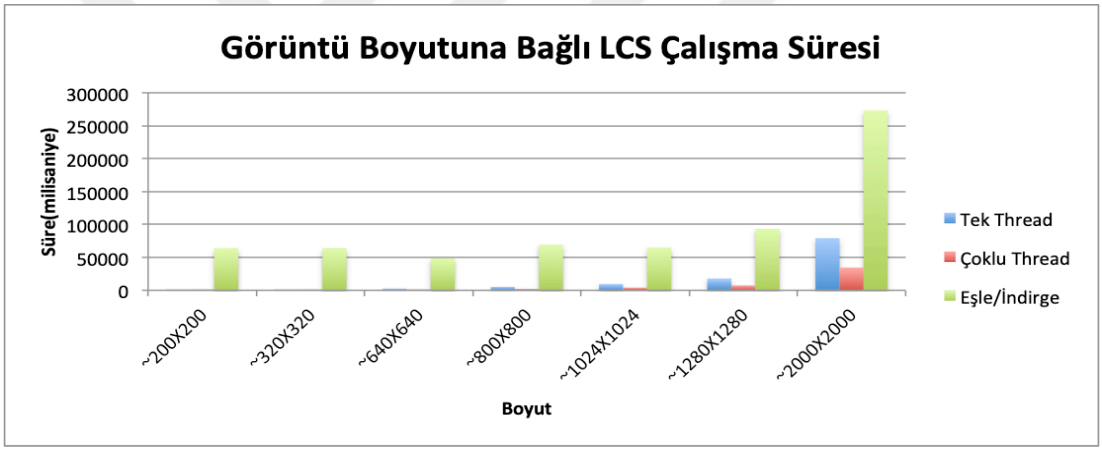


Şekil 4.3. Görüntü Boyutuna Bağlı KK Üretme Süresi

Üç farklı uygulama görüntü boyutuna bağlı LCS çalışma süreleri Tablo 4.3 ve Şekil 4.1'deki gibidir.

Tablo 4.3. Görüntü Boyutuna Bağlı LCS Çalışma Süresi

	Tek Thread (milisaniye)	Çoklu Thread (3 Thread, milisaniye)	Eşle/İndirge (milisaniye)
~200X200	81	54	63821
~320X320	208	155	63965
~640X640	2579	1187	48556
~800X800	5176	2147	69177
~1024X1024	9490	4152	64589
~1280X1280	17798	7579	93304
~2000X2000	79035	34542	273091

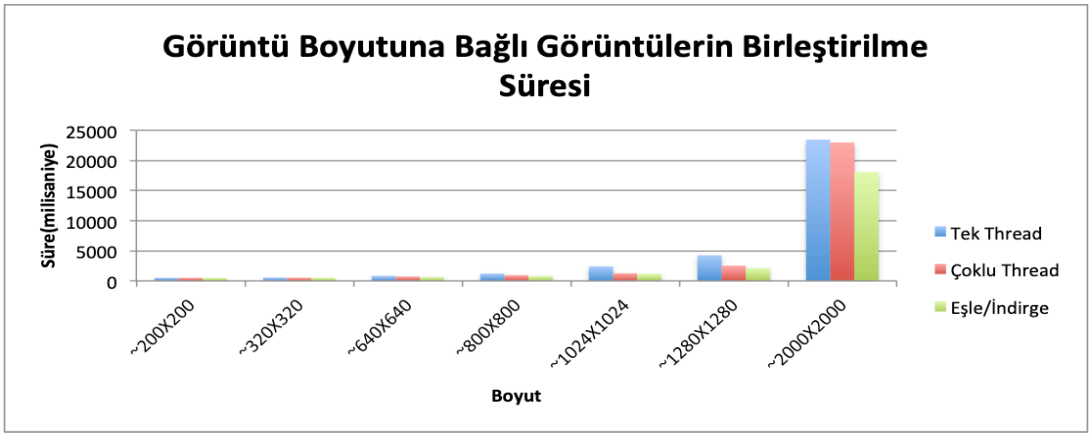


Şekil 4.4. Görüntü Boyutuna Bağlı LCS Çalışma Süresi

Üç farklı uygulamada LCS sonuçlarına göre görüntülerin boyutuna bağlı birleştirilme süreleri Tablo 4.4 ve Şekil 4.3'deki gibidir.

Tablo 4.4. Görüntü Boyutuna Bağlı Görüntülerin Birleştirilme Süresi

	Tek Thread (milisaniye)	Çoklu Thread (3 Thread, milisaniye)	Eşle/İndirge (milisaniye)
~200X200	505	501	497
~320X320	547	535	523
~640X640	832	738	644
~800X800	1201	947	793
~1024X1024	2389	1262	1167
~1280X1280	4242	2526	2103
~2000X2000	23435	22968	18034



Şekil 4.5. Görüntü Boyutuna Bağlı Görüntülerin Birleştirilme Süresi

5. SONUÇLAR VE ÖNERİLER

Yapılan çalışmada resimlerin Hadoop Eşle/İndirge programlama modeliyle birleştirilmesine yönelik algoritmalar geliştirilmeye çalışılmıştır. Resimlerin Eşle/İndirge programlama modelinde birleştirile bilmesi için resimlerin pikselleri kullanılarak resimler metin formatına çevrilmiş ve resimlere ait metinler LCS algoritmasıyla karşılaştırılıp resimlerin nereden birleştirileceği bulunmuştur. Resimleri işlemek için Hadoop ortamının kullanılmasında büyük resimlerin için ölçeklenebilir bir sistem oluşturulması hedeflenmiştir.

Çalışma sürecinde resimlerin birleştirilmesi için geliştirilen algoritmanın ve çözüm yönteminin Hadoop Eşle/İndirge mimarisine uyarlanması ve performans çıktılarının değerlendirile bilmesi için java programla dilinde tek makinede tek thread ve çoklu thread olarak çalışacak şekilde geliştirmeleri yapılmıştır.

Geliştirilen bu sistemlerde sentetik verilerle görüntülerin birleştirilmesine yönelin testler yapılmıştır. Yapılan testler neticesinde küçük ölçekli görüntülerin birleştirilmesi tek makinede çalışan çözümlerde daha performanslı olduğu gözlemlendi. Büyük ölçekli resimlerin birleştirilmesinde ise tek makinada geliştirilen çözümün çalışmadığı ve Hadoop ortamı için geliştirilen çözümün ise başarılı sonuç dönmese de çalıştığı gözlemlenmiştir.

Geliştirilen sistemler kaynak kullanımı, genişletile bilirliği (scalability) ve yönetim yönünden karşılaştırttığımızda; tek makinede geliştirilen çözümlerin kaynak kullanımı uygulamanın çalıştırıldığı makineyle sınırlıdır, Apache Hadoop ortamları için geliştirilen çözümde böyle bir kısıtlama yoktur. Yönetimsel olarak karşılaştırdığımızda Apache Hadoop ortamlarının yönetimi ve işletimi tek makineye göre daha karmaşıktır.

Görüntüleri birleştirmek için yapılan geliştirmeler ve testler önerilen çözümün Hadoop Eşle/İndirge programlama modeline uygulana bilirliğini gösterse de geliştirilen yöntemde kullanılan 3 farklı Eşle/İndirge işi ve üç makineli bir Hadoop

kümesinde çalışması nedeniyle istenilen performans çıktıları elde edilememiştir. Algoritmanın bir sonraki adımında geliştirilen sistemin daha az sayıda Eşle/İndirge işi ile yapılması ve daha büyük ölçekli Hadoop kümesinde çalıştırılması ve büyük veri işlem araçlarından olan Apache Spark kullanılarak geliştirilmesi hedeflenmektedir.

Geliştirilen bu yöntemle büyük boyutlu resimlerin Hadoop ortamlarında birleştirile bileceğinin gösterilmiştir. Aynı zamanda görüntü birleştirme kullanılarak yapılan panoramik görüntü oluşturma için Hadoop ortamlarının kullanılabileceğini göstermiştir.

KAYNAKLAR

[1] Doğan K., Arslantekin K., Büyük Veri: Önemi, Yapısı Ve Günümüzdeki Durum, *DTCF Dergisi*, 2016, **56**(1), 15-36.

[2] https://www.zarantech.com/blog/the-4-vs-of-big-data/, (Ziyaret Tarihi: 20 Kasım 2018).

[3] White T., *Hadoop: The Definitive Guide*, 5rd ed., O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA, 2015.

[4] Shvachko K., Kuang, H. Radia, S. Chansler, R., The Hadoop Distributed File System, *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, Incline Village, NV, USA, 3-7 May 2010.

[5] Kim Y., Araragi, T. Nakmura, J. Masuzawa T., A Distributed NameNode Cluster for a Highly-Available Hadoop Distributed File System, *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*, Nara, Japan, 6-9 Oct. 2014.

[6] https://wiki.apache.org/hadoop/NameNode, (Ziyaret Tarihi: 19 Kasım 2018).

[7] https://www.cloudera.com/documentation/enterprise, (Ziyaret Tarihi: 19 Kasım 2018).

[8] https://wiki.apache.org/hadoop/DataNode, (Ziyaret Tarihi: 19 Kasım 2018).

[9] https://www.cloudera.com/documentation/enterprise, (Ziyaret Tarihi: 19 Kasım 2018).

[10] Borthakur D., *HDFS Architecture Guide*, Copyright © 2008 The Apache Software Foundation.

[11] Lammel R., Google's MapReduce programming model, *Science of Computer Programming*, 2018, **70**(1), 1-30.

[12] Yang Y., Liu Z., Fu Y., MapReduce as a programming model for association rules algorithm on Hadoop, *The 3rd International Conference on Information Sciences and Interaction Sciences*, Chengdu, China, 23-25 June 2010.

[13] Dean J., Ghema S., MapReduce A Flexible Data Processing Tool, *Communications of the ACM*, 2010, **53**(1), 72-77.

[14] Kalavri V., Vlassov V., MapReduce: Limitations, Optimizations and Open Issues, *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, Melbourne, VIC, Australia, 16-18 July 2013.

[15] https://wiki.apache.org/hadoop/JobTracker, (Ziyaret Tarihi: 19 Kasım 2018).

[16] https://wiki.apache.org/hadoop/TaskTracker, (Ziyaret Tarihi: 19 Kasım 2018).

[17] https://www.journaldev.com, (Ziyaret Tarihi: 20 Aralık 2018).

[18] http://hipi.cs.virginia.edu, (Ziyaret Tarihi: 15 Aralık 2018).

[19] Arsh S., Bhatt A., Kumar P., Distributed image processing using Hadoop and HIPI, *International Conference on Advances in Computing, Communications and Informatics (ICACCI),* Jaipur, India, 21-24 Sept. 2016.

[20] https://en.wikipedia.org/wiki/Image_registration, (Ziyaret Tarihi: 17 Aralık 2018).

[21] Lucas D., Kanade T., An Iterative Image Registration Technique with an Application to Stereo Vision, *Proc 7th Int. Joint Conf. on Artificial Intelligence(IJCAI)*, Vancouver, British Columbia, Canada, 24-28 August 1981.

[22] Dowson N., Bowden R., Mutual Information for Lucas-Kanade Tracking (MILK): An Inverse Compositional Formulation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008, **30**(1), 180-185.

[23] Brown L.G., A survey of image registration techniques. *ACM Computing Surveys*, 1992, **24**, 325-376.

[24] Szeliski R., Image Alignment and Stitching: A Tutorial, *Foundations and Trends in Computer Graphics and Vision*, 2016, **2**(1), 1-104.

[25] Image Stitching techniques By Ryan Maponga, ECE '17, https://sites.tufts.edu/eeseniordesignhandbook/files/2017/05/Yellow_Maponga_F1.pdf, (Ziyaret Tarihi: 20 Aralık 2018).

[26] Li Y., Ma L., A Fast and Robust Image Stitching Algorithm, *6th World Congress on Intelligent Control and Automation*, Dalian, China, 21-23 June 2006.

[27] Huang C., Lin S., Chen J., Efficient Image Stitching of Continuous Image Sequence With Image and Seam Selections, *IEEE Sensors Journal*, 2015, **15**(10), 5910-5918.

[28] Brown, M., Hartley, R., Nister, D., Minimal Solutions for Panoramic Stitching, *IEEE Conference on Computer Vision and Pattern Recognition*, Minneapolis, MN, USA, 17-22 June 2007.

[29] Juan L., Oubong G., SURF applied in panorama image stitching, *2nd International Conference on Image Processing Theory, Tools and Applications*, Paris, France, 1-10 July 2010.

[30] Brown, M, Lowe, D., Automatic Panoramic Image Stitching using Invariant Features, *International Journal of Computer Vision*, 2007, **74**(1), 59–73.

[31] Sweeney C., Arietta S., and Lawrence J., Liu L., HIPI: A Hadoop Image Processing Interface for Image-based MapReduce Tasks, Undergraduate Thesis, Computer Science, University of Virginia, 2011.

[32] Almeer M.H., Cloud Hadoop Map Reduce For Remote Sensing Image Analysis, *Journel of Emerging Trends in Computing Information Sciences*, 2012, **3**(4), 637–644.

[33] Malakar R., Vydyanathan N., A CUDA-Enabled Hadoop Cluster For Fast Distributed Image Processing, *Parallel Computing Technologies Parcomptech,* Bangalore, India, 21-23 February 2013.

[34] Demir İ., Sayar A., Hadoop Plugin For Distributed And Parallel Image Processing, *Signal Processing and Communications Applications Conference*, Mugla, Turkey, 18-20 April 2012.

[35] Eken S., Sayar A., Vectorization and Spatial Query Architecture on Island Satellite Images*, AWERProcedia Information Technology and Computer Science*, 2012, **2**, 37-43.

[36] Eken S., Sayar A., Performance Evaluations of Vector-Raster Satellite Image Transfers through Web Services, *IEEE 36TH Annual Computer Software and Applications Conference*, Izmir, Turkey, July 2012.

[37] Eken S., Sayar A., Mert Ü., Registering LandSat-8 Mosaic Images: A Case Study on Marmara Sea, *10th International Conference on Electronics, Computer and Computation,* Ankara, Turkey, 7-9 Nov. 2013.

[38] Eken S., Sayar A., An Automated Technique to Determine Spatiotemporal Changes in Satellite Island Images with Vectorization and Spatial Queries, *Sadhana - Academy Proceedings in Engineering Science*, DOI: 10.1007/s12046-014-0309-7, **40**(1), 121–137.

[39] Eken S., Sayar A., A MapReduce based Big Data Framework for Object Extraction from Mosaic Satellite Images, *International Conference on Internet of Things and Big Data (Doctoral Consortium)*, Roma, Italy, 23-25 April 2016.

[40] Eken S., Sayar A., Uydu Görüntülerinin Yüksek Performansta İşlenmesi Üzerine Bir İnceme: Vektör Tabanli Mozaik Örme Durum Çalışması, *6. Uzaktan Algılama ve CBS Sempozyumu*, Adana, Türkiye, 5-7 Ekim 2016.

[41] Eken S., Aydın E., Sayar A., HIPI Kullanarak Çok sayıda Raster Uydu Görüntüsünün Dağıtık Mimaride Vektörleştirilmesi, *International Conference on Artificial Intelligence and Data Processing*, Malatya, Turkey, 16-17 Sept. 2017.

[42] Markonis D., Schaer R., Eggle I., Müller H., Depeursingle A., Using MapReduce for Large-Scale Medical Image Analysis, *2012 IEEE Second International Conference on Healthcare Informatics, Imaging and Systems Biology*, San Diego, CA, USA, 27-28 Sept. 2012.

[43] Ryu C., Lee D., Jang M., Kim C., Seo E., Extensible Video Processing Framework in Apache Hadoop, *5th International Conference on Cloud Computing Technology and Science*, Bristol, UK, 2-5 Dec. 2013.

[44] Yamamoto M. and Kaneko K., Parallel image database processing with mapreduce and performance evaluation in pseudo distributed mode, *International Journal of Electronic Commerce Studies*, 2012, **3**(2), 211-228.

[45] Sozykin A., Epanchintsev T., MIPr - a framework for distributed image processing using Hadoop, *9th International Conference on Application of Information and Communication Technologies*, Rostov on Don, Russia, 14-16 Oct. 2015.

[46] Rajak R., Raveendran D., Chandrasekhar Bh. M., Medasani S.S., High Resolution Satellite Image Processing Using Hadoop Framework, *International Conference on Cloud Computing in Emerging Markets*, Bangalore, India, 25-27 Nov. 2015.

[47] Banaei S. ve Moghaddan H., Hadoop and Its Role in Modern Image Processing, Open Journal of Marine Science, 2014, **4**, 239-245.

[48] Kocakulak H. ve Taşkaya T., A Hadoop solution for ballistic image analysis and recognition, *International Conference on High Performance Computing & Simulation*, Istanbul, Turkey, 4-8 July 2011.

[49] https://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Longest_common_substring, (Ziyaret Tarihi: 15 Aralık 2018).

[50] Tseng, C., Yang, C., Ann, H., Efficient Algorithms for the Longest Common Subsequence Problem with Sequential Substring Constraints, *11th International Conference on Bioinformatics and Bioengineering*, Taichung, Taiwan, 24-26 Oct. 2011.

[51] https://en.wikipedia.org/wiki/Longest_common_subsequence_problem, (Ziyaret Tarihi: 15 Aralık 2018).

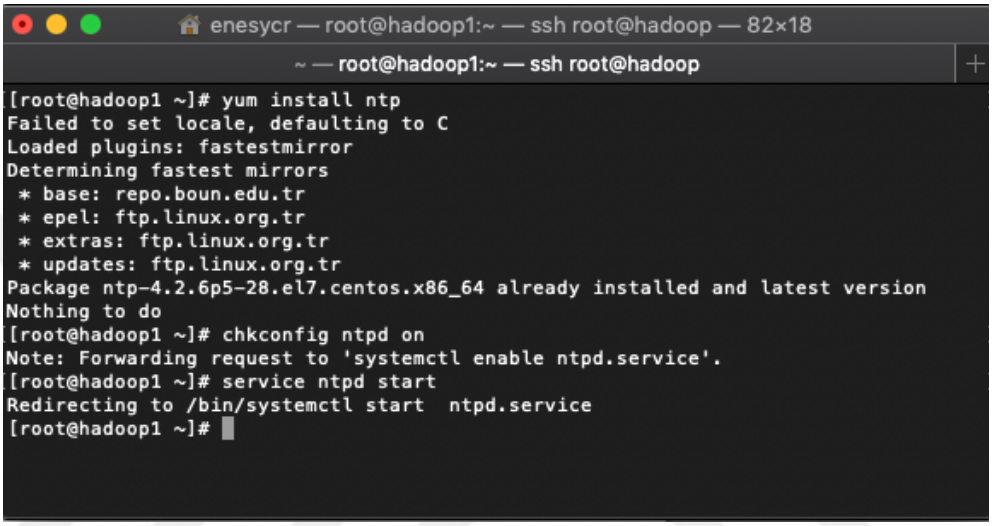
[52] Liu J., Wu, S., Research on longest common subsequence fast algorithm, *International Conference on Consumer Electronics, Communications and Networks*, XianNing, China, 16-18 April 2011.

EKLER

EK-A

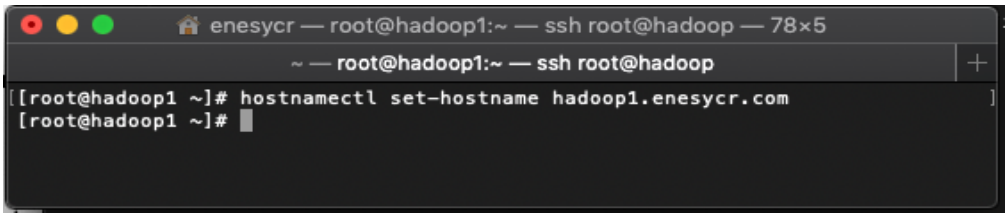
Cloudera Hadoop Kurulumu

- Hadoop Cloudera sürümü kurumumu için 3 tane sanal veya fiziksel makineye Centos 7 kurularak network ayarları yapılmıştır.
- Makinelerin tarih senkronizasyonu için Şekil A.1' de gösterin komutlar terminalde çalıştırılıp ntp kurumumu ve ayarları yapılmıştır.



Şekil A.1. Centos ntp kurulumu

- Cluster içindeki bütün makinelerin hostname atamak için Şekil A.2'deki komut çalıştırılmıştır. Aynı zamanda Tablo A.1'de belirtilen clusterdaki bütün makineler ait hostname ve ip bilgileri "/etc/hosts" dosyasına eklenmiştir.

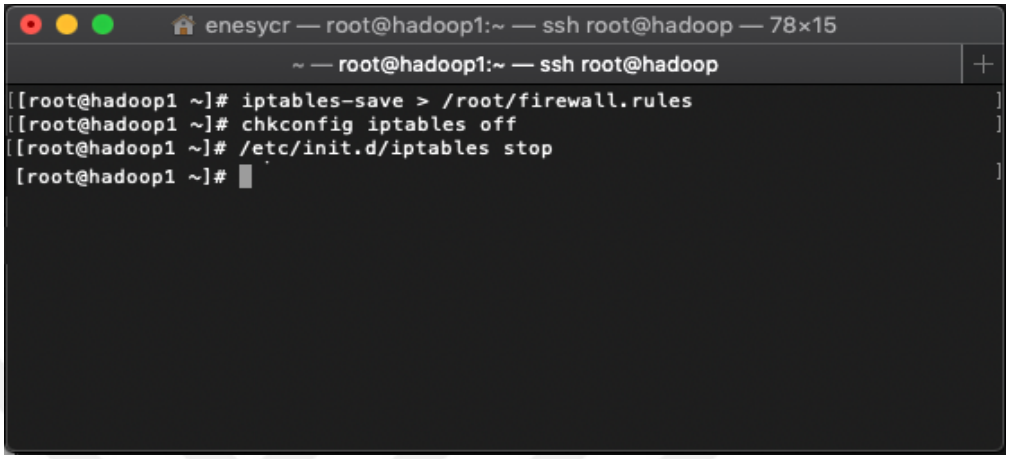


Şekil A.2. Terminal üzerinden hostname atama

Tablo A.1. Bilinen hostname listesi

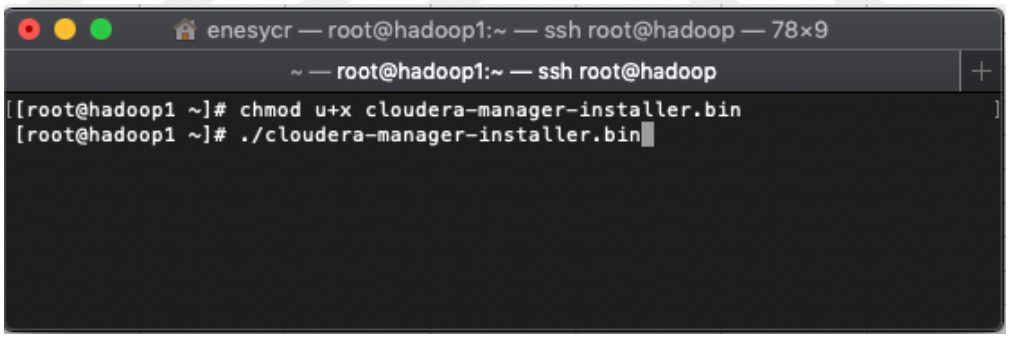
10.1.10.35 hadoop1.enesycr.com hadoop1
10.1.10.36 hadoop2.enesycr.com hadoop2
10.1.10.37 hadoop3.enesycr.com hadoop3

- “/etc/selinux/config” dosyası açılmıştır “SELINUX=enforcing” değer “SELINUX=disables” olarak değiştirilerek Selinux iptal edilmiştir.
- Firewall kaydedilerek kapatılmıştır. Bu işlem için Şekil A.3’deki komutlar çalıştırılmıştır.



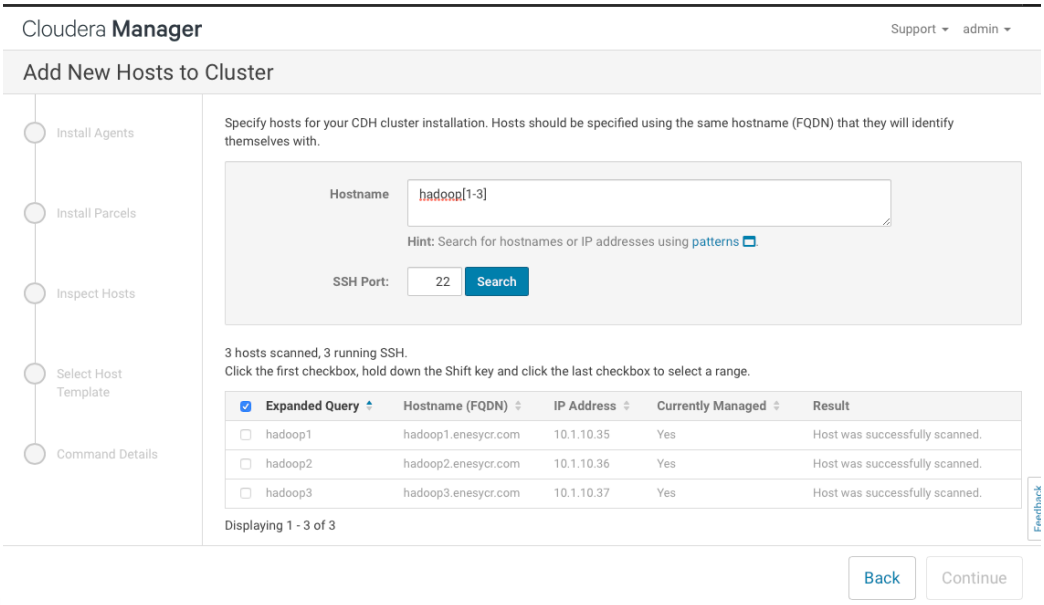
Şekil A.3. Terminalden firewall kapatma

- Cloudera manager installer cloudera sitesinden indirilerek Şekil A.4’de gösterildiği gibi sahiplik ayarı yapılır.

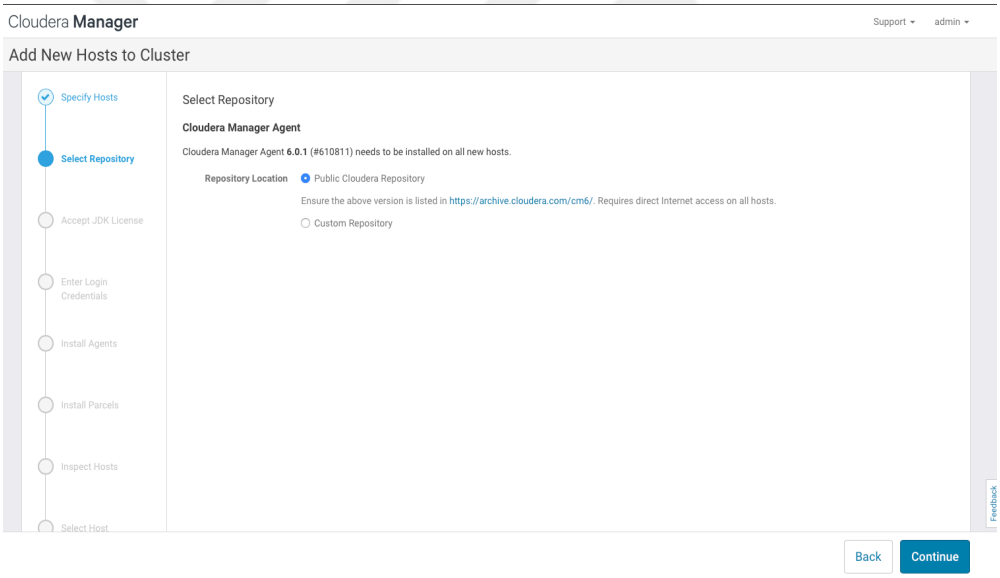


Şekil A.4. Terminalden CM sahiplik ayarı yapma ve CM'yi çalıştırma komutu

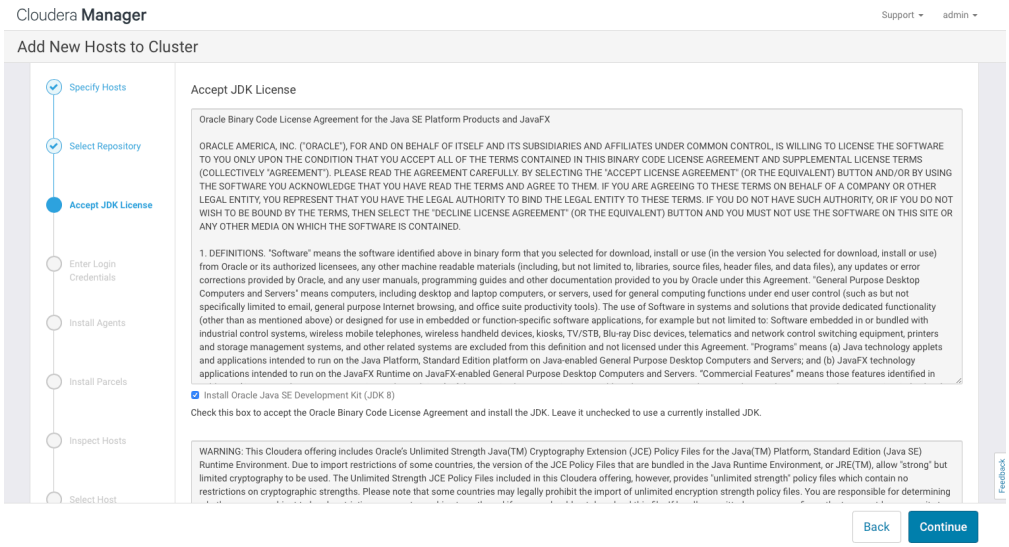
- CM kurulumu yapıldıktan sonra “cloudera_manager_host:8180” adresine gidilir, hadoop kurulumu yapılacak makinelerin hostları Şekil A.5’te gösterildiği gibi bulunur ve hostlara CM agent kurulumuyla birlikte java (Şekil A.6, Şekil A.7, Şekil A.8 bk.), hadoop kurulumu yapılır(Şekil A.9’a bk.). Şekil A.10’a gösterilen hadoop bileşenleri CM arayüzü üzerinden eklenir.



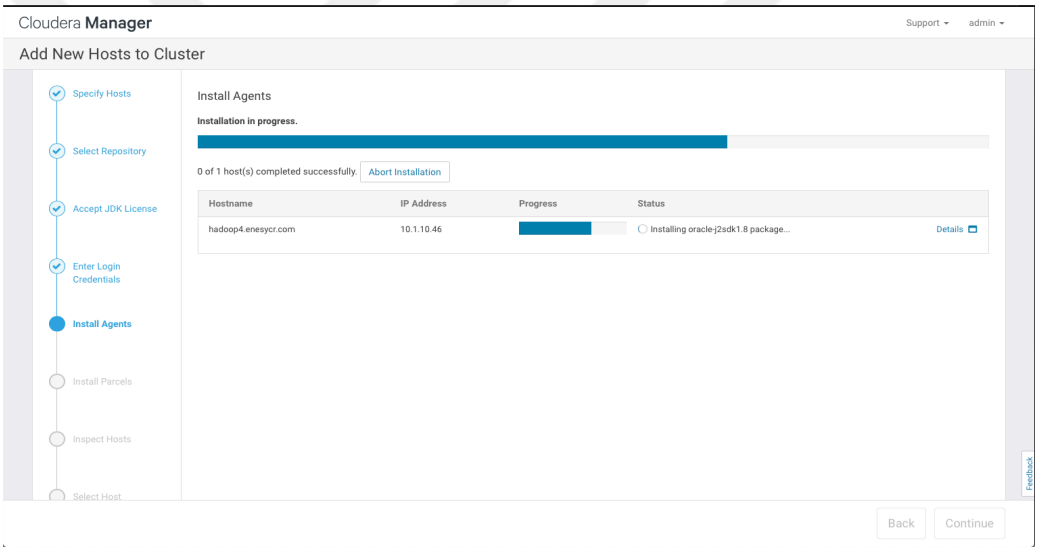
Şekil A.5. CM ara yüzü üzerinden hadoop kurulumu yapılacak makinelerin hostlarının bulunması



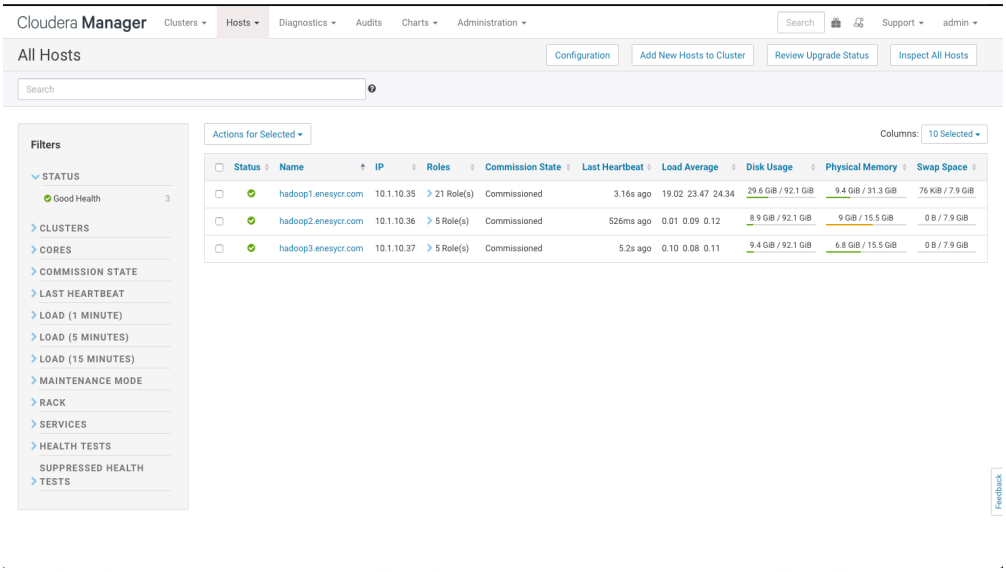
Şekil A.6. CM arayüzü üzerinden cloudera repository seçimi



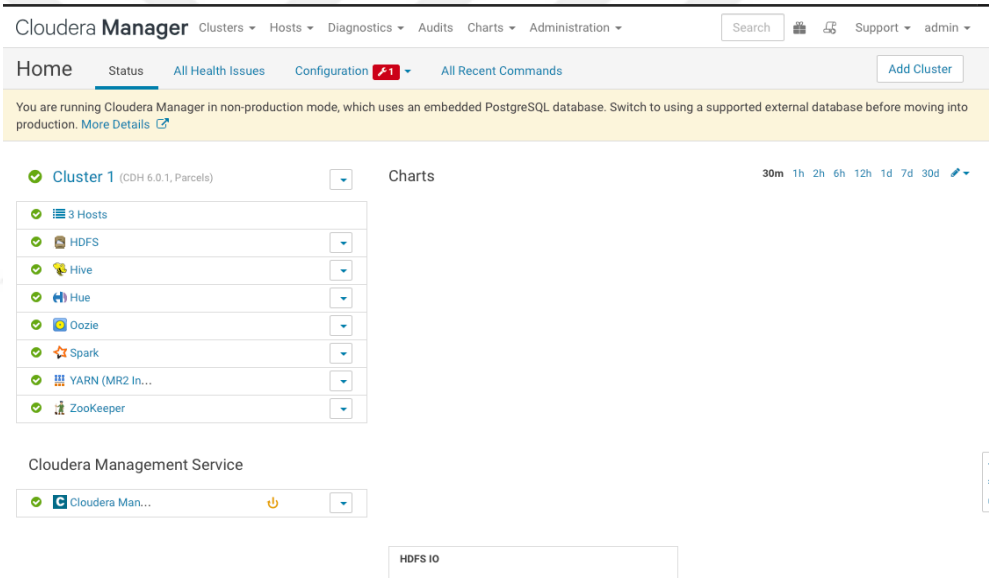
Şekil A.7. CM arayüzü üzerinden makinelere kurulacak java lisans kabulü



Şekil A.8. CM arayüzü üzerinden cloudera ajan, java ve hadoop kurulum süreci



Şekil A.9. CM arayüzü üzerinden hadoop kurulumu yapılmış host bilgileri



Şekil A.10. CM arayüzü üzerinden eklenmiş hadoop bileşenleri

KİŞİSEL YAYINLAR VE ESERLER

Yücer E., Sayar A., Eken S., Distributed Image Matching With Longest Common Subsequence Algorithm, *International Marmara Science And Social Science Congreass (IMASCON 2018)* , Kocaeli, Turkey , 23-25 November 2018.

ÖZGEÇMİŞ

Enes Yücer 1988'de Tokat'ta doğdu. Lise öğrenimini Tokat Mehmet Akif Ersoy Lisesi'nde tamamladı. 2006 yılında girdiği Dokuz Eylül Üniversitesi Bilgisayar Mühendisliği Bölümü'nden 2012 yılında mezun oldu. 2017 yılında Kocaeli Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı'nda yüksek lisans eğitimine başladı. Yüksek lisans eğitiminde hadoop eşle/indirge mimarisi kullanılarak görüntülerin en uzun ortak alt dizi algoritmasıyla örülmesi yönelik çalışmalarda bulundu. Ayrıca, 2016 yılında TÜBİTAK BİLGEM'de başladığı Araştırmacı görevini halen sürdürmektedir.