İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BÖLÜMLEME ALGORİTMALARI İLE VERİ TEKİLLEŞTİRME

YÜKSEK LİSANS TEZİ

Duygu COŞGUN

Matematik Mühendisliği Anabilim Dalı

Matematik Mühendisliği Yüksek Lisans Programı

HAZİRAN 2018

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BÖLÜMLEME ALGORİTMALARI İLE VERİ TEKİLLEŞTİRME

YÜKSEK LİSANS TEZİ

Duygu COŞGUN

509131064

Matematik Mühendisliği Anabilim Dalı

Matematik Mühendisliği Yüksek Lisans Programı

Tez Danışmanı: Dr. Öğr. Üyesi Bahri GÜLDOĞAN

HAZİRAN 2018

İTÜ, Fen Bilimleri Enstitüsü'nün 509131064 numaralı Yüksek Lisans Öğrencisi Duygu COŞGUN, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "BÖLÜMLEME ALGORİTMALARI İLE VERİ TEKİLLEŞTİRME" başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı : **Dr. Öğr. Üyesi Bahri GÜLDOĞAN**
İstanbul Teknik Üniversitesi

Jüri Üyeleri : **Prof. Dr. Fatma ÖZDEMİR** .....
İstanbul Teknik Üniversitesi

Dr. Öğr. Üyesi Alp Arslan BAYRAKÇİ .....
Gebze Teknik Üniversitesi

Teslim Tarihi : 03 Mayıs 2018
Savunma Tarihi : 04 Haziran 2018

Aileme,

ÖNSÖZ

Yüksek lisans ve bitirme tezim boyunca her zaman yanımda olan ve desteklerini esirgemeyen saygıdeğer danışman hocalarım Prof.Dr. Fatma ÖZDEMİR ve Dr. Öğr. Üyesi Bahri GÜLDOĞAN'a teşekkürlerimi sunarım. Çalışmalarım boyunca benden bir an olsun yardımlarını esirgemeyen nişanlım Volkan AKŞAN'a ve tüm zorlukları benimle göğüsleyen ve hayatımın her evresinde bana destek olan annem Dudu COŞGUN, babam Hüseyin COŞGUN ve kardeşim Mustafa COŞGUN'a sonsuz teşekkürlerimi sunarım.

Mayıs 2018 Duygu COŞGUN

İÇİNDEKİLER

Sayfa

KISALTMALAR

CD-ROM **:** Compact Disk Read Only Memory
DVD-ROM **:** Digital Versatile Disk
SD **:** Secure Digital
BSW **:** Basic Sliding Window Algorithm
TTTD **:** Two Thresholds Two Divisors Algorithm
MOBESE **:** MOBil Elektronik Sistem Entegrasyonu
FTP **:** File Transfer Protokol

SEMBOLLER

Δ	**:** Referans verilen ve farklılık gösteren veriler ile oluşturulan değer
W	**:** Bölümleme algoritmasında kullanılan pencere boyutu
D	**:** Bölümleme algoritmasında kullanılan tam sayı böleni
R	**:** Bölümleme algoritmasında kullanılan tam sayı kalanı
P	**:** Bölümleme algoritmasında kullanılan konum belirleyici parametre
h	**:** Özet değer
wC	**:** Zayıf Sağlama Toplamı
sC	**:** Güçlü Sağlama Toplamı

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

BÖLÜMLEME ALGORİTMALARI İLE VERİ TEKİLLEŞTİRME

ÖZET

Teknolojinin ilerlemesi ve kullanım alanlarının artması ile birlikte günümüzde bilginin gücü ön plana çıkmakta ve değer kazanmaktadır. Bilgilerin yapılandırılıp kayıt altına alınması ile veriler meydana gelmektedir. Veriye olan ihtiyaç arttıkça, veri için gerekli olan işlem sayısı ve depolama için gerekli olan kapasite kullanım alanının da hızla artmasına sebep olmaktadır. Hem yazılımsal hem de donanımsal olarak bazı çözümler üretmek çağımızın önemli araştırma konularından biri haline gelmiştir.

Veri depolama, bilginin saklanmasını, aynı zamanda korunmasını ve çeşitlilik gösteren verilerin sınıflandırılarak ulaşılabilir olmasını sağlar. Böylelikle verinin yönetimi kolaylaşır ve bilgiye hızlı ve güvenli bir şekilde erişimi sağlar. Bu alanda günümüzde karşılaşılan önemli sorunlardan biri, depolanacak olan kaynaklardan çok daha fazla veri üretilmesidir.

Aynı verinin farklı dosyalarda birden fazla kez bulunması ile veri tekrarı meydana gelmektedir, bu da sistemin yavaşlamasına ve hard diskte fazla yer kaplamasına neden olur. Ek olarak birden fazla dosyada bulunmasıyla, veri içeriğinin bir dosyada güncellenip diğerlerinde güncellenmemesi durumunda veri bütünlüğünün bozulmasına sebep olabilmektedir. Bu çalışma kapsamında depolama sistemlerinde veri tekrarlılığını önlemek ve verilerin tekilleştirilmesi için geliştirilen bazı algoritmalar incelenmiştir.

Büyük ölçekli veri yedeklemelerinden dolayı yüksek maliyetler meydana gelmektedir. Bu maliyetleri düşürmek ise günümüzde önemi artmakta olan araştırma konularından biridir. Bu kapsamda tekrarlanan, dağınık ve kirli verilerin standardizasyonunun sağlanarak temizlenmesi için veri tekilleştirme yöntemleri geliştirilmiştir.

Veri tekilleştirme temel olarak üç adımdan oluşmaktadır. Bu adımların birincisi dosya bölümleme işlemleri ile başlamaktadır ve çalışma kapsamında bölümleme kavramının ve kullanılan algoritmaların detaylarına yer verilecektir. Dosya bölümleme işlemi yapıldıktan sonra ise özet değer üretme adımına geçilmektedir. Bu aşama bloklara ayrılan her parçanın temsil edileceği değerler hesaplanmasından oluşmaktadır. Bu adım tamamlandıktan sonra ise son olarak artıklık tespiti gerçekleştirilir ve bu sayede tekrar eden verilerin ayıklanması ve sadece depolama sistemine yeni ulaşan verilerin saklanması için gerekli işlemler gerçekleştirilir.

Tez kapsamında öncelikle yukarıda özetlenmiş olan veri tekilleştirmenin tanımlanması ile kullanıldığı lokasyona bağlı yöntemler ve veriyi işleme açısından kullanılan teknikler ele alınacaktır. Tekilleştirme işlemlerinin en önemli ve zaman alan kısmı olarak bölümleme kavramı detaylı olarak incelenecektir. Bu bağlamda tekilleştirme ile depolama alanından tasarruf edilebilmesi için blok ve dosya bazında

bölümleme işlemlerinin nasıl uygulanıldığından bahsedilecektir. Bununla ilgili kullanılan BSW ve TTTD algoritmalarının işleyişi incelenerek, algoritmaların avantaj ve dezavantajları değerlendirilikten sonra, alternatif çözümlere yer verilecektir. Seknronizasyon işlemlerinde kullanılan önemli algoritmaların işleyişi incelenerek bölümleme çalışmaların bir uygulamasının bulunduğu senkronizasyon aracı olan Syncany ile örneklendirilmesi sağlanacaktır. Örneklendirmeler farklı işletim sistemlerine sahip aygıt ile çeşitli dosyaların senkronizasyonunun sağlanması ve bu senkronizasyon işleminin örnek olarak istemci tarafında gerçekleştirilmesi ile gözlemlenecektir.

DATA DEDUPLICATION WITH CHUNKING ALGORITHMS

SUMMARY

The power of knowledge plays a significant role with the advancement of technology and the increased number of implementations. By configuration and storage of the information, data sets are constructed. The need of data sets increases the number of transactions and the storage capacity for data. Therefore, hardware and software based approaches for the solution of the problems became the popular research topics of today's scientific world.

The main aim of data storage is the data safety and classification of the variety of stored data types. Thus, the data management makes data accessibility faster and more secure. One of the main issues of the field is the generation of redundant datasets requiring much more storage resources.

Duplicates in the data files cause the system slowness and redundant space on the hard disk. The updates of only the specific data files in the whole dataset may disrupt data integrity due to the duplicates in the files. This study aims to prevent data repetition in storage systems and some algorithms developed for deduplication of data are examined.

Data deduplication basically consists of three steps. The first of these steps begins with file partitioning and details of the partitioning concept and the algorithms used will be included in the study. After file partitioning is done, the step of generating summary value is started. This step consists of calculating the values to be represented for each part separated into blocks. After this step is completed, the redundancy detection is finally performed and the necessary steps are taken in order to sort out the repeated data and only to store the data newly arriving in the storage system. In data backup and data storage systems, data deduplication methods are being applied to reduce network bandwidth and consequently to keep the network less busy, to provide faster and more efficient data exchange. The deduplication technology detects identical files in the case where the same file is uploaded by multiple clients in the storage area, so that only one copy is stored. This would save the high cost of large-scale data backup operations in storage costs. For example; mail attachments sent to more than one user, duplication of the same documents for different users' personal use, etc. cause unnecessary use of disk space. Here, a single copy of the solution is kept and a logical reference is created so that users can access the database when needed. The motion vector method used for compressing and storing video recordings on mobile cameras is one of the common examples used in the field of deduction to detect motion and change. Chunking is the process of searching redundancy detection and partitioning a file. Each file part is defined as chunks, and chunking is the most time-consuming part of data deduplication systems

because a file is scanned from the beginning. Therefore, the way in which chunking algorithms are handled on the file is the determining factor.

In principal, major storage units are hard disks, memory cards, compact flash (computer and digital flash memory type used in cameras), secure digital cards (SD card), optical readers (CD / DVD) and the cloud storage systems can be accessed via computers and mobile devices.

The hard disks mentioned above are the storage resources used to record big programs and dataset for further usages. In general, hard disks are used as the fixed hardwares of computers. On the hard disks, the information is backed up in zip disks with almost 100 MB capacity. Memory cards, which allow storage of data in removable media, and reduces the usage of internal memory cards which are the main limitations of the data storage. For example compact flash drives are the most common and old storage cards, called also flash cards varying from 8 MB to 1 GB. SD cards are the smallest storage units which can be usable for a variety of devices. On the other hand, USB Flash Drives with high quality performance, small size, and plug-and-play feature are the most commonly preferred storage units providing the fastest and easiest way to store data. CD-ROM drives; named also optical readers, provide high-capacity and quality of data storage with almost 7,800 KB data flow per second. These drives are widely preferred for the data types;such as sound, picture, video, requiring great amount of spaces for the storage. DVD-Rom has the same physical dimensions and features as the CD-ROM; however, has much higher capacity of storage area.

Since the cloud storage systems came into our lives, the drives and data storage tool that has been used for a long time, have been replaced with cloud systems which can be accessed by internet using any fixed or portable products. Moreover, it is possible to access cloud systems via the internet through tablets or smartphones. The well-known cloud storage systems are Google Drive, Dropbox, Amazon Cloud, G Cloud, and Apple ICloud.

Due to the fact that the big data storage methods cost a lot, reducing the cost of the data storage is one of the popular research topics with increasing importance. Therefore, data deduplication have been developed for the standardization of repetitive, dispersed and dirty data sets. Within the scope of this thesis, first the data uniqueness, its applied methods, then how the storage space was saved with data uniquification and, block and file-based chunking, are described. With the analysis of the related algorithms, BSW and TTTD, and their applications provided in a synchronized tool called Syncany, the algorithms and the methods are exemplified.

In the thesis, first of all, the definition of data deduplication summarized above and the methods used in terms of location processing and data processing will be discussed. The concept of chunking will be examined in detail as the most important and time consuming part of the deduplication process. In this context, it will be explained how block and file-based chunking are applied so that deduplication can save storage space. We will examine the operation of the BSW and TTTD algorithms, and discuss the advantages and disadvantages of the algorithms and then discuss alternative solutions. By studying the operation of the important algorithms used in the synchronization processes, it will be exemplified by Syncany, the synchronization tool in which an application of the chunking works is found. Sampling will be handled by synchronizing the various files with the device having

different operating systems, and by performing this synchronization on the client side as an example.

When the results obtained are examined, the use of multiple segments to reduce network bandwidth and prevent unnecessary use of disk space plays an important role in achieving targeted data deduplication methods. In order to ensure that the same files are transmitted to the remote data storage area, data exchange is kept at minimum level, since it is sufficient to provide only the index transmission. In addition to this, making improvements on Syncany in order to increase the sensitivity to deduplication in the synchronization of the changes made on the file will lead to more efficient results.

1. GİRİŞ

Bilgilerin saklanması, korunması ve sınıflandırılması için kullanılan teknolojiler veri depolama olarak isimlendirilmektedir. Bu teknolojiler, veri kayıplarını önler ve bilgi akışını ve verilerin güncellenmesini hızlı ve yüksek erişilebilirlik ile sağlar. Günümüzde Bilgi Teknolojisi firmalarının yatırım maliyetlerinde tasarruf ve kazanım sağlayan, bilgileri verimli kaynaklara dönüştüren sistemlerdir.

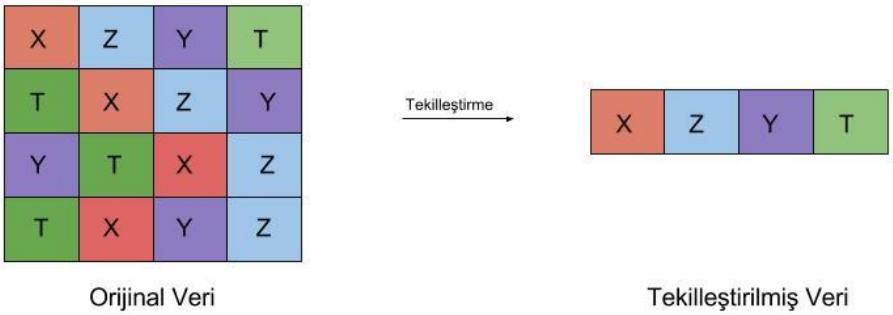
Başlıca depolama birimleri; sabit disk, bellek kartları, compact flash (bilgisayar ve dijital fotoğraf makinelerinde kullanılan flash bellek çeşidi), secure digital (SD Kart), optik okuyucular (CD/DVD) ve bulut depolama türünde internet erişimi vasıtasıyla kullanıcıların bilgisayar veya mobil aygıttan istediği zaman ulaşabildiği saklama alanlarıdır. Yukarıda bahsedilen sabit disk, büyük hacimlerdeki program ve verilerin kaydedilip daha sonra kullanılabilir olduğu depolama alanlarıdır. Genellikle bilgisayar kasaları içerisinde sabit olarak kullanılan sabit disk, hard disk olarak da isimlendirilmektedir. Sabit disk üzerindeki bilgileri yaklaşık 100 MB kapasiteye sahip olan zip disk sürücüleri yedeklemektedir. Bellek kartları ise, verilerin depolanarak çıkarılabilir ortamlarda saklanmasını sağlayan ve dâhili belleklerin kullanımını azaltarak sınırlandırılmasını engelleyen araçlardır. Örneğin compact flash denilen en yaygın ve eski teknoloji kartları 8 MB ve 1 GB arasında değişebilen kart teknolojisidir. SD Kartlar ise çeşitli aygıtlara uygun olabilen en küçük depolama birimlerindendir. Bunların yanında en yaygın kullanılan USB Flash Disk'ler ise yüksek performansları, küçük boyutları ve takıp çalıştırılma özelliği ile dijital olarak dokümanların aktarımını sağlamasıyla en hızlı ve kolay kullanım yöntemini sunmaktadır. Optik okuyucular olarak isimlendirilen CD-Rom sürücüleri, yaklaşık olarak saniyede 7.800 KB veri akışını sağlayan plastik yüzeyli ve yüksek kapasiteli veri depolama birimidir. Ses, resim, video gibi çok büyük yer kaplayan yazılımlar için yaygın olarak kullanılmaktadır. DVD-Rom sürücüleri ise, CD-Rom'a göre fiziksel olarak aynı ölçülere sahip olmasına rağmen saklama alanı olarak çok daha yüksek kapasiteye sahiptir.

Bulut depolama teknolojilerinin hayatımıza girmesiyle, uzun zamandır kullanılan bu yöntemlerin yerine ağ bağlantısı aracılığıyla erişim sağlanabilen sistemler geliştirilmiştir. Herhangi bir sabit veya taşınabilir ürünü kullanma gereksinimi olmaksızın depolanan veriler erişilebilir hale gelmiştir. Üstelik bilgisayarların yanında tablet veya akıllı telefonlar aracılığı ile internet erişiminin olduğu her noktada ulaşılabilir olmuştur. En bilinen bulut depolama sistemleri, Google Drive, Dropbox, Amazon Cloud, G Cloud, Apple ICloud olarak gösterilebilir.

2. VERİ TEKİLLEŞTİRME NEDİR?

Veri yedekleme ve veri depolama sistemlerinde, ağ bandı genişliğini düşürmek ve dolayısıyla ağı daha az derecede meşgul etmek, daha hızlı ve verimli veri alışverişini sağlamak amacıyla veri tekilleştirme yöntemleri uygulanmaktadır. Tekilleştirme teknolojisi, birden fazla istemci tarafından depolama alanına aynı dosya yüklendiği durumda özdeş dosyaları algılayarak yalnızca bir kopyasının saklanmasını sağlar. Bu sayede depolama maliyetlerinde büyük ölçekli veri yedekleme işlemlerinde ortaya çıkan yüksek maliyetten tasarruf sağlanabilecektir. [1]

Örneğin; birden fazla kullanıcıya gönderilmiş mail eklentileri, aynı dokümanların farklı kullanıcıların kişisel kullanımları için tekrar kopyalanması gibi işlemler disk alanının gereksiz kullanımına sebep olur. Burada çözüm verinin tek bir kopyasının tutulması ile kullanıcıların gerektiğinde veriye ulaşabilmesi için mantıksal bir referans yaratılmasıdır (Şekil 2.1). Mobese kameralarındaki video kayıtların sıkıştırılarak depolanmasında kullanılan hareket vektör (motion vector) yöntemi, hareketi ve değişikliği algılamak için tekilleştirme alanında kullanılan yaygın örneklerden biridir.



Şekil 2.1 : Orjinal ve tekilleştirilmiş veri örneği

Veri tekilleştirme yöntemlerinin avantajları:

1. Disk alanı ve donanım ihtiyaçlarının azalmasını sağlar. Bu sayede enerji tüketimi ve ekipman bakımı azaltılır.

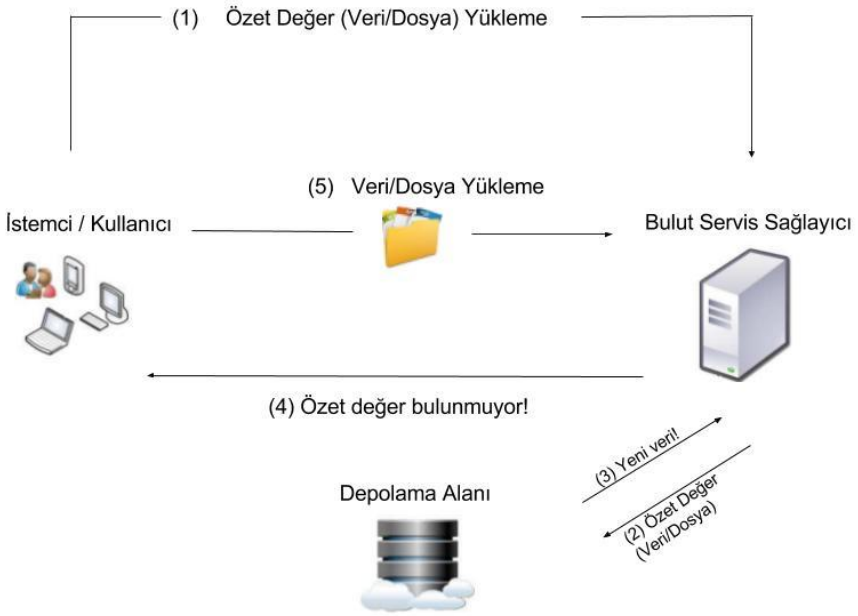
2. Depo alanı kullanımından ortaya çıkan maliyetleri düşürür.
3. Gereksiz veya tekrarlayan verileri kaldırarak veri aktarımlarını optimal seviyede tutar.
4. Üzerinde çalışılması gereken verinin azaltılması sayesinde bölümleme gibi işlemlerin daha hızlı yürütülmesini sağlar.
5. Veri yedekleme ve veri kurtarma verimi artar.
6. Güncellemelerin kontrol ve takip edilebilmesine olanak sağlar.

2.1 Veri Tekilleştirme Yöntemleri

Veri tekilleştirme teknikleri, tekilleştirmenin yapıldığı lokasyona bağlı olarak iki metot ile sınıflandırılabilir:

2.1.1 İstemci tarafında (client-side) veri tekilleştirmesi

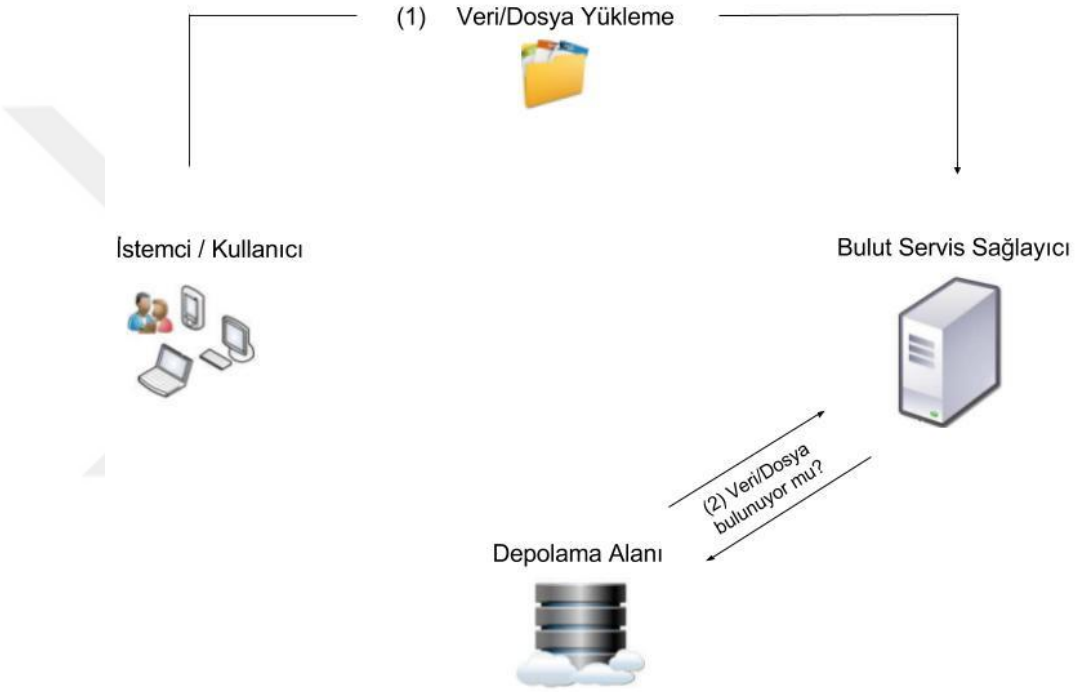
İstemci tarafında veri tekilleştirmesiyle veriler üzerindeki tekrarlılık kontrolü kullanıcıların dosyaları yüklemesinden önce, istemci tarafında gerçekleşir. Özet değerler yardımıyla dosyanın tümü yerine daha küçük boyutlu veriler iletilerek kontroller sağlanır. İletilen özet değer depolama alanında bulunmuyor ise, dosya veya verinin sunucuya gönderilmesi sağlanır (Şekil 2.2). [2] Burada tekilleştirme işlemi istemci tarafında yapıldığından, yüksek bant genişliği maliyetleri azaltılmış olur.



Şekil 2.2 : İstemci tarafında veri tekilleştirme akışı.

2.1.2 Sunucu tarafında (server-side) veri tekilleştirmesi

Sunucu taraflı tekilleştirme bir sunucunun iki kayıtlı dosyanın aynı olup olmadığını kontrol etmesini sağlar. Burada dosyanın kendisi istemci tarafından veri tekilleştirilmesi yapılmadan sunucuya gönderilir, daha sonra sunucu verinin veya dosyanın tekrarlılığını kontrol eder (Şekil 2.3). [2] Bu durumda istemci tarafında verilerin yinelenen veri olup olmadığı tespit edilemez, dolayısıyla bant genişliği azalmaz. Sadece sunucu için depolama maliyeti azalmış olur. Bu bağlamda istemci tarafında veri tekilleştirme yöntemi daha verimlidir.



Şekil 2.3 : Sunucu tarafında veri tekilleştirme akışı.

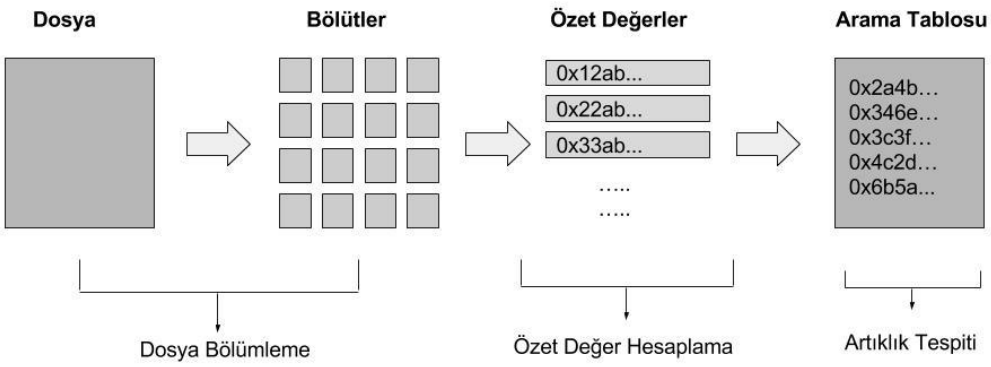
2.2 Veri Tekilleştirme Yaklaşımları

Yukarıda bahsi geçen metotlara ek olarak veri işleme yöntemi açısından veri tekilleştirme teknikleri ikiye ayrılır:

2.2.1 Özet tabanlı (hash-based) yaklaşım

Bu yaklaşım üç bileşenden oluşmaktadır (Şekil 2.4):

i. Dosya Bölümleme
ii. Özet Değer Üretme
iii. Artıklık Tespiti



Şekil 2.4 : Özet Tabanlı Tekilleştirme Sisteminin Bileşenleri

Yeni bir dosya geldiğinde, veri tekilleştirme sistemi öncelikle dosyayı bölüt ismi verilen küçük bloklar halinde parçalara ayırır. Tez kapsamında bu aşamada kullanılan bölümleme algoritmalarına daha detaylı olarak ilerleyen bölümlerde değineceğiz.

İkinci aşamada ise, bloklar halinde parçalara ayrılan dosyanın her bir bölütün tekil imza olarak temsil edebilmesi için özet değerleri hesaplanır.

Son olarak, elde edilen özet değerler veri tekilleştirme sistemi ile veri tabanı veya arama tablolarında depolanan daha önceden depolama sisteminde tutulan özet değerler ile karşılaştırılır. Herhangi bir eşleşme bulunmadığı durumda, söz konusu bölüt yeni veri olarak kabul edilir ve depolama alanında saklanır. Karşılaştırmalar sırasında bir benzerlik bulunduğunda ise tekrarlanan veri için bir referans oluşturulur. Bu referans veri tabanında depolanmış aynı dosyayı içeren kopyaya oluşturulur. [3]

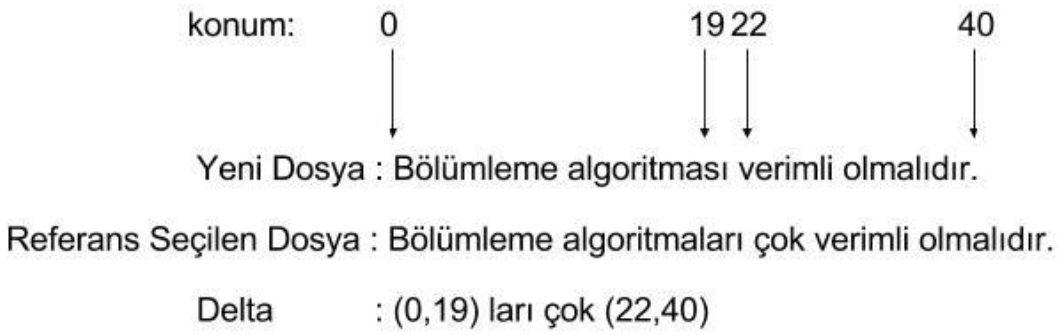
Özet tabanlı yaklaşım avantaj ve dezavantajları:

- Bölütler sabit boyutlu olduğundan, parçalama işlemleri basit ve kolay bir şekilde yapılır.
- Özet tabanlı yaklaşım ile bölütler arasında veri içeriği ve türünden bağımsız olarak karşılaştırma yapıldığından, hangi verilerin ne ile karşılaştırıldığı kontrol edilemez.
- Veri tekrarlılığı olasılığı dikkate alınmaksızın her gelen bölüt ile arama tablosunda yer alan özetler karşılaştırılır, bu nedenle ciddi performans sorunlarına sebep olur.
- Büyük veri hacimlerini etkin bir şekilde işlemesi açısından zayıftır.

2.2.2 İçerik tabanlı (content-aware) yaklaşım

Bu yaklaşımda aşağıdaki adımlar izlenerek tekilleştirme sağlanır:

1. Yeni gelen dosya ile veritabanı üzerindeki veriler üzerinde benzerlikler ve ilişkiler belirlenir. Örneğin; html,pdf,word gibi dosya tipi kıyaslanır.
2. Dosyanın belirli özelliklerine bağlı olarak (dosya uzantısı, adı, boyutu vb.) ortak özellikler bulunuyor ise depolama alanında bulunan bu dosya referans dosyası olarak seçilir.
3. Bölümleme algoritmaları yardımıyla bölütler elde edildikten sonra byte-byte veya blok-blok benzerlik incelemeleri yapılır.
4. Burada bir delta (Δ) (Şekil 2.5) değeri hesaplanarak referans dosyası ve yeni gelen dosya arasındaki farklılık ve benzerlikler bulunur. Sistem sadece (Δ) ve bir gösterge yardımıyla yeni gelen dosya yerine referans dosyayı depolar. [3]



Şekil 2.5 : İçerik tabanlı yaklaşım için bayt bazında karşılaştırma örneği.

İçerik tabanlı yaklaşım avantaj ve dezavantajları:

- Veri içerik ve türüne göre incelendiğinden daha küçük bir veri kümesi üzerinde analiz yapılır.
- Karşılaştırmalar sonucunda sistemde depolanan veri sadece delta ve referans olduğundan aktarılan veri miktarı en aza indirgenir.
- Özet tabanlı yaklaşıma göre daha verimli ve düşük maliyetli bir alternatif sunar.
- Bölüt sınırlarını belirlemek için uygulanacak bölümleme algoritmalarına bağlı olarak performans problemi yaşanabilir.

Yukarıda bahsedilen tekilleştirme yöntemlerinden bağımsız olarak, bölümleme algoritmaları tekilleştirme sistemleri için çok önemli bir role sahiptir. Bu bağlamda,

tez çalışması kapsamında iki farklı bölümleme algoritması üstünde durulacaktır. Bir sonraki bölümde ise bölümleme ile ilgili temel kavramlar incelenecektir. Bölümleme teknikleri ele alınarak çoğunlukla kullanılan yöntemler üzerinde durulacaktır.

3. BÖLÜMLEME NEDİR?

3.1 Temel Kavramlar

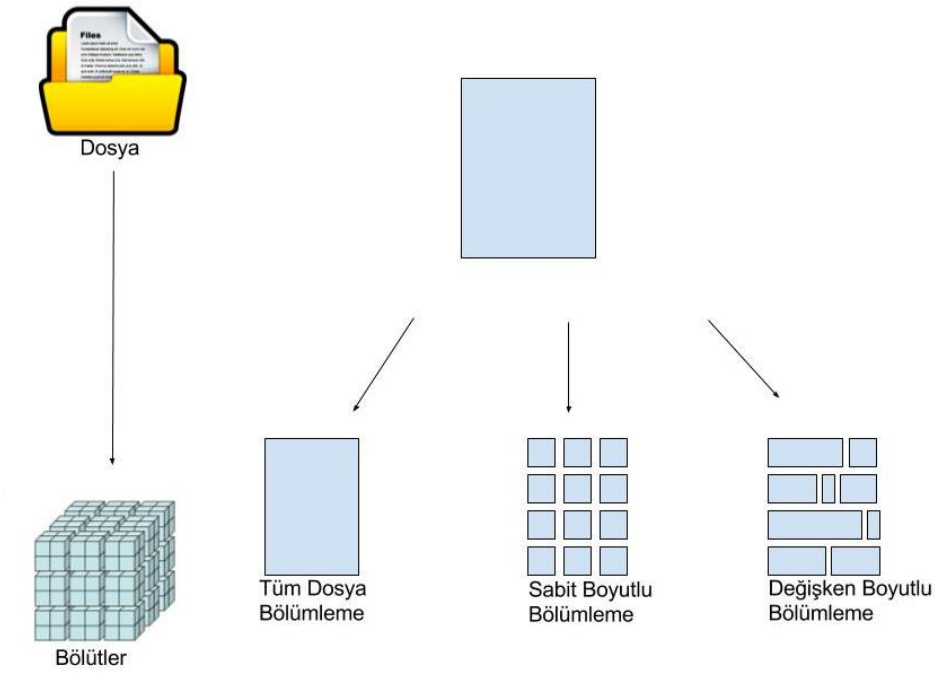
Bölümleme bir dosyanın taranması ve parçalara bölünmesi işlemidir. Her bir dosya parçasına bölüt adı verilir ve bir dosyanın baştan uca tarayarak ayrıştırılmasından dolayı bölümleme işlemleri veri tekilleştirme sistemlerinde en fazla zaman harcayan kısımdır. Dolayısıyla bölümleme algoritmalarının dosya üzerindeki parçalama işlemlerindeki yöntemleri belirleyici faktördür.

Bölüt boyutlarının küçük olması veri tekilleştirme sisteminin iyi sonuç vermesine fayda sağlarken, bölüt sayılarının artmasından dolayı bölütlerin depolandığı arama tablolarının boyutlarının artmasına sebep olur. Sistemin çalışma süresinde artışa sebep olabileceği gibi disk alanı ihtiyacı sebebiyle yüksek maliyetli donanım gereksinimleri oluşabilir. Özetle, bölümleme algoritmaları için aşağıdaki koşulların sağlanması gerekmektedir:

- ✓ İşlem süresinin minimum tutulması
- ✓ Bölüt sayısı ve tekilleştirme işlemlerinin dengeli olacak seviyede gerçeklenmesi
- ✓ Bölüt boyutlarının çeşitliliğinin kontrollü olması

Örneğin, bir dosya içerisinde tekrarlanan veri bulunuyor ise, bu verinin tekrarlı kısmı için olabildiğince büyük bölütler halinde paketlemek kullanışlıdır. Farklılık gösteren kısımlar var ise küçük bölütler halinde depolanabilir. Bu şekilde dosyanın veya verinin niteliklerine uygun olacak şekilde yaygın olarak kullanılan bölümleme yöntemleri 3 kategoriye ayrılır (Şekil 3.1).

3.2 Veri parçalama teknikleri



Şekil 3.1 : Veri Parçalama Teknikleri.

1. Tüm Dosya Bölümleme: Tüm dosya bir bölüt olacak şekilde kullanılır. Bu yönüyle veri parçalama teknikleri arasında en hızlı ve en basit uygulanabilen yöntemdir fakat veri tekilleştirme işleminde başarısızdır.

2. Sabit Boyutlu Bölümleme: Sabit boyutlu bölümleme tekniği, seçilen sabit bir blok boyunu kullanır. Depolanan nesnelerin içeriğinden ve nesnelerden bağımsız olarak bu boyuttaki bloklara bölünürler. [4]

Sabit boyutlu bölümleme, Venti [Quinlan ve Dorwards 2002] ve Oceanstore [Kubiatowicz ve ark.] gibi içerik adresli sistemlerde kullanılır. [5,6]

Bu yöntemde veri tekilleştirmenin başarılı olabilmesi için dosyayı küçük boyutlu bloklara parçalamak daha verimli sonuç vermesini sağlayacaktır, ancak fazla kapasite kullanımına ve veri üzerinde yapılan güncellemelere karşı iki sürüm arasındaki kopyaları eleme işleminde dayanıksızdır. Bu sorun aynı zamanda Sınır Kayma Problemi olarak adlandırılır ve yukarıda bahsedilen her iki veri parçalama yönteminde karşılaşılan en önemli sorundur. Bu problemin nasıl meydana geldiğini ve çözümü için geliştirilen değişken boyutlu bölümleme yöntemlerini daha detaylı olarak inceleyeceğiz.

3. Değişken Boyutlu Bölümleme: Veri güncellemelerinde değişikliklerde tekrarlılık gösteren verilerin tekilleştirilmesine duyarlılık, nesneleri değişken boyutlu parçalara bölerek sağlanabilir. Böylece ardışık sürümlerde yapılan değişiklikler, değişim bölgesi çerçevesinde çok daha az bölütler ile depolanır.

3.2.1 Sınır kayma problemi

Bir dosya üzerindeki veri güncellemeleri sonrasında ardışık sürümler arasındaki tekilleştirme işlemlerinde karşılaşılan bir problemdir. Hem tüm dosya bölümleme hem de sabit boyutlu bölümleme yöntemlerinde meydana gelebilir. Örneğin, bir dosyanın herhangi bir kısmına tek bir bayt eklemek veya çıkarmak, dosyadaki tüm blokların içeriğini değiştirebilir ve mevcut bloklarla karşılaştırıldığında ortaklık payının azalmasına sebep olur. Çünkü blokların içeriği değişeceğinden, buna bağlı olarak farklı özet değerleri üretilecektir. Dolayısıyla bir bayt'lık değişiklik sebebiyle bloklar arası benzerliklerin saptanması zorlaşacaktır. [7] Aşağıdaki (Şekil 2.2) orijinal dosya ve güncellenen dosya örnekleri ile sınır kayma problemini inceleyelim.

Şekil 3.2 : Sınır Kayma Problemi Örneği.

Yukarıdaki şekilde görüldüğü gibi, güncellenen dosyada 2.bölüt içerisinde "12" şeklinde 2 byte'lık veri eklendiğinde, diğer tüm değerler aynı kalsa da 1.bölütten sonraki tüm bölütler içerisindeki değerler değişeceğinden farklı özet değerleri ve dolayısıyla farklı sonuçlar elde edilecektir.

Değişken boyutlu bölümleme yöntemi ile bu sınır kayma probleminden kaçınılabilir. Aynı zamanda bu yöntem ile depolama hizmetleri kullanıcılarının kullandığı alan miktarını önemli ölçüde azaltılabilir. Kullanıcı bir dosya üzerinde değişiklik yaptığında, sabit boyutlu bölme işleminin tersine bölümleme algoritması, dosyanın hangi bölümlerinin değiştirilmediğini algılar ve bunları tekrar aktarmaz.

Bu bölüm itibariyle temel olarak bölümleme ile ilgili kavramlar incelenmiş olup, veri parçalama tekniklerinden yola çıkılarak ele alacağımız algoritmalara temel oluşturulmuştur.

4. BÖLÜMLEME ALGORİTMALARI

4.1 Temel Hareketli Pencere (BSW) Algoritması

Bu algoritmanın temel amacı, sınır kayma probleminden kaçınarak ağ bant genişliği gereksinimlerini azaltmaktır. Bu bağlamda temel hareketli pencere algoritmasının kullanım yöntemini, verimli olduğu yanları ve geliştirilmesi gereken eksik kaldığı durumları inceleyeceğiz.

BSW algoritmasında önceden yapılandırılması gereken ana parametreler bulunmaktadır. Bu parametreler algoritma çalışmadan önce belirlenir, çalışması süresince de bu değerler sabit olarak kullanılır.

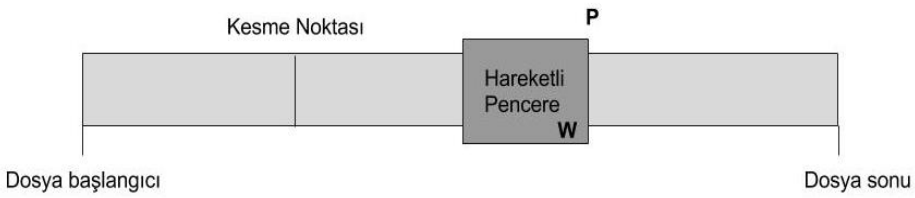
BSW Algoritmasında Kullanılan Ana Parametreler:

1. $W = Pencere\ Boyutu$
2. $D = Tamsayı\ Böleni$
3. $R = Tamsayı\ Kalanı$

Bu değişkenler başlangıçta belirlendikten sonra algoritma çalışması boyunca sabit kalması gerekmektedir. Örneğin, pencere boyutu genel olarak 48 byte olarak seçilmektedir. [8,9] Bu sabit pencere boyutu ile aşağıda bahsedilecek kurallar doğrultusunda tekilleştirme işleminin yapılması için dosya taranacaktır. Diğer parametrelerden tamsayı böleni 1000, tamsayı kalanı ise 0 olarak seçilmektedir.

4.1.1 BSW algoritmasının işleyişi:

Algoritmanın çalışması (Şekil 4.1) ile gösterildiği gibi hareketli pencere ile dosya üzerinde ilerletilerek gerçekleşmektedir.



Şekil 4.1 : BSW Algoritmasının İşleyişi

1.Adım: Sabit boyutlu (W) penceresi dosyanın başından sonuna kadar 1 bayt aralıklarla kaydırılır.

2.Adım: Pencerenin son noktasını P ile isimlendirirsek, her P konumu için W penceresi içerisinde kalan verilerin özet değerleri bulunur. Bu özet değer Rabin Fingerprint Algoritması ile hesaplanmaktadır. Bulunan özet değeri h olarak isimlendirelim.

3.Adım:

$$h \bmod D = R \tag{4.1}$$

(4.1) eşitliği bulunduğu noktada, p konumu kesme noktası olarak belirlenir. Böylece W kayar penceresi p konumundan başlayacak şekilde kaydırılır, hesaplama ve karşılaştırmalar tekrar edilir.

4.Adım:

$$h \bmod D \neq R \tag{4.2}$$

(4.2) eşitliği sağlandığı durumda ise kayar pencere 1 bayt kaydırılır, hesaplama ve karşılaştırmalara dosya sonuna kadar bu şekilde devam edilir.

BSW algoritması, dosyanın içeriği ile birlikte bölüt sınırlarını belirlediğinden, bu yaklaşımın sınır kaydırma probleminden kaçınmak için başarılı olduğu kanıtlanmıştır.

Bu algoritmada D parametresi, bölüt boyutlarının belirlenebilmesi için önceden seçilen sayıdır. R ise 0 ile D-1 arasındaki bir sayıdır. Her konum için hesaplanan özet değerler ile bu seçilmiş sayılar arasında modüler işlemler uygulanarak bölütlerin boyutlarına karar verilir. Dolayısıyla bu parametrelerden D sayısı, en önemli rolü oynamaktadır. Çünkü bölütlerin boyutlarını beklentimize en yakın şekilde yapılandırmayı sağlar.

W kayar penceresinin her 1 bayt kaydırılması ile (4.1) eşitliğinin sağlanabilmesi olasılığı $\frac{1}{D}$ 'dir. Dolayısıyla her D bayt'ta bir bölüt sınırını belirlemek için kesme noktasının bulunması beklenebilir.

Örneğin, D değerini 100 olarak seçersek ve her parçanın boyutunu 100 bayt olarak değerlendirirsek, R değeri ise,

$$0 \leq R \leq 99 \tag{4.3}$$

(4.3) eşitsizliği aralığında olan herhangi bir tam sayıdır. Dolayısıyla her 100 kaydırma neticesinde en azından bir tane (4.1) denkleminin sağlanmasını bekleriz. Bu da kesme noktasını belirleyen bir ölçüt olduğu için, bölüt boyutunun 100 bayt olabilmesine tekabül eder. Bir başka deyişle bölüt boyutu 100 bayt ile sınırlanmış olacaktır.

Kayar pencere içerisinde kalan alanın özet değeri h olarak isimlendirilmişti. Bu değer $k - bit$ boyutta olup 0 ve 1'lerden oluşan sayıdır. Hesaplanmasında ise, Rabin Fingerprint Algoritması kullanılır.

4.2 Rabin parmak izi algoritması:

Parmak izleri, daha büyük nesneler için kısa etiketlerdir. İki parmak izi farklıysa, karşılık gelen nesneler de kesinlikle farklıdır ve iki farklı nesnenin aynı parmak izine sahip olması olasılığı oldukça küçüktür. Bu nedenle veriler arasındaki benzerlik oranını azaltmak için parmak izi yöntemi ile etiket olarak isimlendirdiğimiz özet değerleri bulunmaktadır.

Bir diğer deyişle, parmak izi şeması belirli bir fonksiyon koleksiyonudur: [10]

$$F = \{ f : \Omega \rightarrow \{0,1\}^k \} \tag{4.4}$$

Burada Ω olası tüm ilgili nesnelerin kümesidir, k ise parmak izi uzunluğudur. Eğer n farklı nesneden oluşan $S \subset \Omega$ kümesini ve rastgele bir $f \in F$ fonksiyonu seçersek,

$$f(A) \neq f(B) \Longrightarrow A \neq B \tag{4.5}$$

olacaktır.

$$f(A) = f(B) \tag{4.6}$$

olduğu durumda ise

$$A \neq B \tag{4.7}$$

olma ihtimali çok düşüktür.

Rabin parmak izi, sonlu bir cisim üzerinde polinomları kullanarak özet değer bulma yöntemidir. Sonlu cisim aritmetiğine dayanır. Diziler için Micheal Oser Rabin tarafından önerilen parmak izi algoritması aşağıdaki şekilde kullanılmaktadır:

$$A = (a_1, a_2, \dots, a_m) \tag{4.8}$$

ikili karakter dizisi olmak üzere, $a_i \in \{0,1\}$ şeklinde tanımlanan elemanlardan oluşan karakterler olsun. [10]

A dizesi , $A(t)$ polinomu ile ilişkilendirilir. Bu polinom;

$$A(t) = a_1 t^{m-1} + a_2 t^{m-2} + \cdots + a_m \tag{4.9}$$

ile tanımlanan ve derecesi $m-1$, katsayıları Z_2 'den olacak şekilde $a_1 \neq 0$ 'dır. $P(t)$ polinomu Z_2 de tanımlı ve derecesi k olan indirgenemez bir polinom olsun. A dizesinin parmak izi aşağıdaki gibidir:

$$f(A) = A(t) mod\ P(t) \tag{4.10}$$

Buradaki indirgenemez polinom, tanımlı olduğu sonlu cisim üzerinde iki farklı polinomun çarpımı olarak yazılamayan polinomlar olarak tanımlanır. Örneğin, Z_2 'de (Çizelge 3.1) ile gösterilen polinomlar indirgenemezdir.

Çizelge 4.1: Derecelerine göre indirgenemez polinomlar.

Derece	İndirgenemez Polinom
1	$x, 1+x$
2	$1+x+x^2$
3	$1+x+x^3$, $1+x^2+x^3$
4	$1+x+x^4$, $1+x^3+x^4$, $1+x+x^2+x^3+x^4$

Rabin parmak izi yöntemine göre 2.dereceden indirgenemez polinom seçerek konuyu örneklendirelim:

$$W = (1,0,1,1) \tag{4.11}$$

olacak şekilde kayar pencere içerisinde kalan karakter dizesi olsun.

Bu durumda $W(t)$aşağıdaki şekilde olacaktır:

$$W(t) = t^3 + t + 1 \tag{4.12}$$

İndirgenemez polinom ise Z_2'de

$$P(t) = t^2 + t + 1 \tag{4.13}$$

olarak seçilsin.

Yukarıda bahsedildiği gibi parmak izi,

$$f(W) = W(t)\ mod\ P(t) \tag{4.14}$$

denklemi ile bulunur.

Burada

$$t^3 + t + 1 \tag{4.15}$$

Polinomunun

$$t^2 + t + 1 \tag{4.16}$$

polinomuna bölünmesinden elde edilen bölüm $t - 1$, kalan ise t'dir.

$f(W)$ polinomunun katsayıları ile benzer şekilde parmak izi bulunan karakter dizesi elde edilir:

$$f(W) = t \rightarrow W' = (0,0,1,0) \tag{4.17}$$

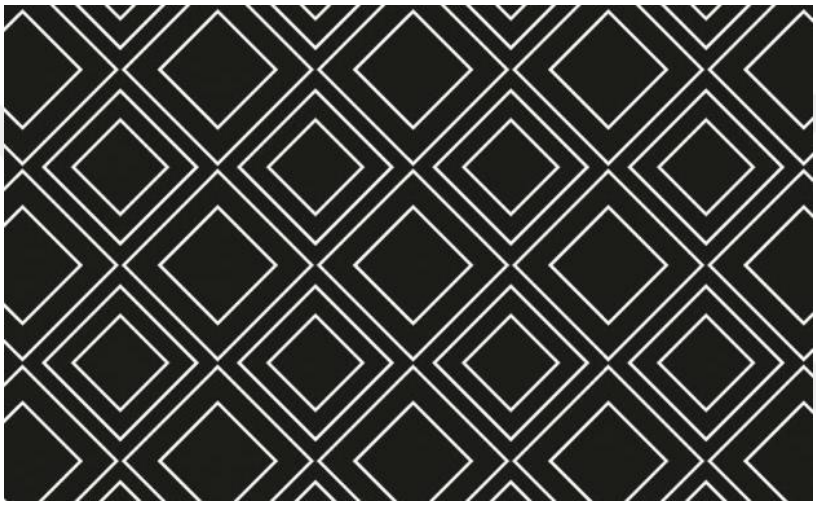
BSW algoritması için kullanılan algoritma ve yöntemler incelendiğinde bu algoritma için iki temel problem bulunmaktadır:

- Birinci sorun, dosya içerisinde birbirini tekrar eden karakter dizeleri içeriyor ise, kayan pencerenin her bir kaydırmada kesme noktası belirleme ihtimali bulunmaktadır.
- İkinci sorun ise kayar pencere tüm dosyayı geçtikten sonra herhangi bir kesme noktası bulamayabilir.

Bu durumda BSW algoritması, tüm dosyayı bir blok olarak ele alabilir, bu da sınır kayma problemine sebep olacaktır. Başka bir deyişle, bölüt boyutlarının çok büyük veya çok küçük olarak bulunmasına neden olabilir.

Bu problemlerin gerçekleşmesi yüksek ihtimalli olmasa da, asla gerçekleşmeyeceğini söyleyemeyiz. Dolayısıyla sadece 1 baytlık değişikliğe bağlı olarak çok küçük veya çok büyük verileri iletmek verimli değildir.

(Şekil 4.2) ile verilen görsel veri üzerinden karşılaşılması olası problemi örnekleyelim. Belirlenen kayar pencere boyutu W iken, içerisinde kalan verinin özet değeri h olsun. (4.1) eşitliğinin sağlanması durumunda kesme noktası belirlendiğinden, bu eşitliğin ilk blokta sağlanması durumunda, 1 bayt kaydırılarak işlem tekrar edileceğinden her blok üzerinde bu şekilde kesme noktası belirlenmesi riski vardır. Tersine, eşitlik sağlanmadığı durumda her 1 bayt'lık kaydırma sonucu oluşan bloklar değişmeyeceğinden kesme noktasının bulunmama durumu oluşacaktır.



Şekil 4.2 : Benzerlikler veri içeren dosya örneği.

Bu problemleri önlemek amacıyla HP (Hewlett-Packard) laboratuarı tarafından 2005 yılında yeni bir algoritma önerilmiştir. Bir sonraki bölümde bu algoritmanın nasıl uygulandığını, avantaj ve dezavantajlarını inceleyeceğiz.

4.3 İki Eşik İki Bölen Algoritması (TTTD)

HP laboratuarı tarafından tasarlanan bu algoritma, rastgele olmayan veriler ile bilinen algoritmalara göre önemli ölçüde daha iyi performans gösteren bölümleme algoritmasıdır. Çıkış noktası BSW algoritmasına dayanmaktadır, ancak farklı olarak iki eşik değeri ve fazladan bir bölen değeri daha kullanılmakta ve bölüt boyutu bu sayede sınırlandırılmaktadır. Böylelikle BSW algoritmasında karşılaşılan yukarıda bahsettiğimiz iki probleme çözüm getirilmektedir. [11]

Üst eşik ve alt eşik değerleri ile bölüt boyları sınırlandırmaktadır. Ayrıca Ana Bölen değeri ile istenen bölüt boyunu belirleyebilmek, aksi takdirde İkincil Bölen ile aday

kesme noktaları belirleyebilmeyi amaçlar. Genel olarak ikincil bölen değeri ana bölen değerinin yarısı olarak belirlenir.

Araştırmalara göre, bölüt boyutu 1000 bayt olarak beklendiği durumda TTTD algoritması için kullanılan dört parametrenin alabileceği uygun değerler aşağıdaki gibidir:

TTTD Algoritmasında Kullanılan Ana Parametreler:

Çizelge 4.2 : TTTD algoritması parametre değerleri tablosu.

Parametre	Kullanım Amacı	Uygun Değerler
Üst Eşik	Bölüt boyutuna üst sınır belirler.	2800 bayt
Alt Eşik	Bölüt boyutuna alt sınır belirler.	460 bayt
Ana Bölen	Kesme noktası belirler.	540
İkincil Bölen	Kesme noktası adayını belirler.	270

Burada ikincil bölen, ana bölen ile kesme noktası bulunmasa bile aday kesme noktası bulunabilmesi için ihtimal sağlar. Kesme noktası bulunması yöntemi BSW algoritmasına benzer şekilde sağlanmaktadır. TTTD algoritmasının dayandığı adımları kullandığı Pseudo kodu yardımıyla aşağıdaki şekilde sıralayabiliriz:

1. Algoritma veri tekilleştirme yapılacak doküman üzerinde her seferinde 1 bayt ilerler ve pencere içinde kalan karakterlerin özetini alır.
2. Eğer şu anki konumla kesme noktası arasında kalan kısmın boyu alt eşikten büyükse, yeni kesme noktasına karar vermek için ana bölen ve ikincil bölen değerleri kullanılır.
3. Üst eşiğe ulaşmadan önce ana bölen değeri ile kesme noktası bulunduysa bölüt sınırlandırılmış olur. Kayan pencere bu yeni kesme noktasından başlar ve hesaplama ve karşılaştırmalar yeni kesme noktası yani bölüt sınırı bulunana kadar tekrarlanır.
4. Algoritma üst eşiğe ulaşırsa, eğer varsa en son bulunan aday kesme noktasına göre bölütün sınırını belirler, aksi taktirde sınır üst eşik olarak belirlenir ve böylece bölütün boyu üst eşik kadar olur.

TTTD algoritmasının yukarıda bahsettiğimiz adımlarını, C kodu ile yazılmış versiyonu ile daha detaylı olarak inceleyeceğiz. Öncelikle kod içerisinde geçen ifadeleri tanımlayalım: [11]

- MinE = Minimum Eşik Değeri
- MaxE = Maksimum Eşik Değeri
- B1 = Ana Bölen
- B2 = İkincil Bölen
- input = pencere içerisinde kalan veri dizisi
- endOfFile (input) = girdi dizisinin sonuna ulaştığında bilgi veren fonksiyon
- getNextByte (input) = bir sonraki baytı getiren fonksiyon
- updateHash (c) = kayar pencere içerisindeki verinin özet değerini alan fonksiyon
- addBreakpoint (p) = Kesme noktası ekleyen fonksiyon
- p = güncel konumu
- l = en son kesme noktası konumu

```
int p = 0, l = 0, backupBreak = 0;

for (; !endOfFile (input) ; p++) {
    unsigned char c = getNextByte (input);
    unsigned int hash = updateHash (c);

    if (p - l < MinE){
       continue;
    }
    if ((hash % B2 ) == B2 - 1){
       backupBreak = p;
    }
    if ((hash % B1 ) == B1 - 1){
       addBreakpoint(p);
       backupBreak = 0;
       l = p;
       continue;
    }
    if (p - l < MaxE){
       continue;
    }
    if (backupBreak != 0){
       addBreakpoint(backupBreak);
       l = backupBreak;
       backupBreak = 0;
    }
    else{
       addBreakpoint(p);
       l = p;
       backupBreak = 0;
    }
 }
 }
```

Şekil 4.3 : TTTD Algoritması Pseudo Kodu.

TTTD Algoritması Akışı:

(1) Başlangıç değerleri belirlenir.

(2) Kayar pencere, bir döngü içerisinde her seferinde 1 bayt kaydırılacak şekilde kullanılır.

(3) Her konum için başlangıç ve bitiş noktasının uzaklığı kontrol edilir. Minimum eşik değerinden küçük ise geri dönülür, 2.adım tekrarlanır. Minimum eşik değerinden büyük ise devam edilir.

(4) $$h = \frac{D}{2} - 1 \left(mod\ \frac{D}{2}\right) \quad (4.18)$$

(4.18) eşitliği sağlandığı durumda bulunulan konum noktası aday kesme noktası olarak belirlenir. Bu eşitlik sağlanmaz ise adımlara devam edilir.

(5) $$h = D - 1(mod\ D) \quad (4.19)$$

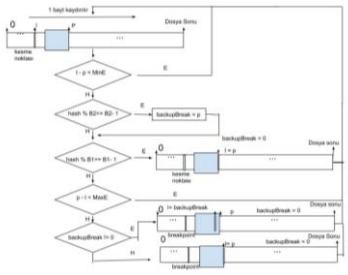
(4.19) eşitliği sağlandığı durumda bulunulan konum noktası kesme noktası olarak belirlenir, aday kesme noktası ise sıfırlanır. Bu eşitlik sağlanmaz ise adımlara devam edilir.

(6) Başlangıç noktası ile bitiş noktasının uzaklığı kontrol edilir. Maksimum eşikten küçük ise adımlara devam edilir. Maksimum eşikten büyük ise geri dönülür.

(7) Bu adıma gelindiğinde, kesme noktası bulunmamıştır. Aday kesme noktası var ise kesme noktası olarak atanır. Güncel konum ise bu aday noktasına getirilir, işlemler bu noktadan itibaren tekrar başlayacaktır. Kesme noktası sıfırlanarak devam edilir.

(8) Aday kesme noktası ve kesme noktası bulunamadığında bu adıma gelinecektir. Bulunulan son konum kesme noktası olarak belirlenir. Kesme noktası bu adımın sonunda sıfırlanarak, bulunulan konum itibariyle döngüdeki işlemlere devam edilir.

Yukarıda bahsedilen TTTD algoritmasına ait akış diyagramı aşağıdaki şekildedir:



Şekil 4.4 : TTTD Algoritması Akış Diyagramı.

TTTD Algoritmasının Avantajları:

1. Minimum ve maksimum eşik değerleri ile çok büyük veya çok küçük bölüt değerlerinin oluşmasını sınırlandırmaktadır.
2. Ana bölen değeri ise, beklenen bölüt boyutlarına yakın değerler elde edilmesini sağlar.
3. İkincil bölen değeri, kesme noktası bulunamadığı durumda aday kesme noktası belirlenmesine yardımcı olur.
4. Yukarıda bahsedilen özellikleri sayesinde, TTTD algoritması ile veri tekilleştirme işlemlerinde zamandan ve depolamadan BSW algoritmasına göre fayda sağlanmış olur. Çünkü bölüt sayıları ve boyutları sınırlandırılmış olacaktır.

TTTD Algoritmasının Dezavantajları:

1. TTTD Algoritması bölüt boyutunu üstten sınırlandırmaktadır. Bu ise çok büyük dosyalarda bölüt sayısının fazla olmasına neden olur. Bölüt sayısının fazla olması bölütlerin bilgilerinin tutulduğu arama tablosunun boyutunu büyütür, böylece arama tablosunda veriler arasında yapılacak karşılaştırmalar için tarama yapılmasının süresi de artacaktır.
2. Diğer bir problem ise, birincil bölenin maksimum eşiğe kadar kesme noktası bulamaması ancak ikincil bölen ile aday nokta bulabildiği halde karşılaştırma ve hesaplama işlemlerinin önceki kesme noktası ile şimdiki konum arasındaki fark maksimum eşik oluncaya kadar devam etmesidir. Bu problemde aday noktalar bulunduğu halde fazladan işlem yapılmaktadır.
3. Ayrıca ikincil bölen ile bulunan aday noktalardan en son bulunanı dikkate alındığından bölüt boyu maksimum eşiğe yakın olabilir.
4. Dosya boyutu; pencere boyutu ve minimum eşikten küçük olduğunda,
   Son kalan parçanın boyutu pencere boyutu ve minimum eşikten küçük olduğunda veya maksimum eşikten büyük olduğunda bölüt algoritma tarafından tespit edilememektedir.

Faydaları ve dezavantajları incelendiğinde, incelediğimiz algoritmalar arasında BSW algoritmasına göre TTTD daha verimli olmasına karşın veri tekilleştirme işlemlerinde bazı eksiklikleri gerçekleşmesi yüksek ihtimalde olmasa da

içermektedir. Bu kapsamda, iyileştirme olarak TTTD-S adı verilen bir algoritma ile ufak bir değişiklik eklenerek çözüm getirilmeye çalışılmıştır. [3]

4.4 TTTD-S Algoritması:

```
1  int p=0, l=0,backupBreak=0;
2
3  for (;!endOfFile(input);p++){
4     unsigned char c=getNextByte(input);
5     unsigned int hash=updateHash(c);
6
7     if (p - l < MinE){
8        continue;
9     }
10    if (p-l > switchP)
11    {
12       switchDivisor();
13    }
14    if ((hash % B2)==B2-1){
15       backupBreak=p;
16    }
17    if ((hash % B1) == B1-1){
18       addBreakpoint(p);
19       backupBreak=0;
20       l=p;
21       resetDivisor();
22       continue;
23    }
24    if ( p-l < MaxE){
25       continue;
26    }
27    if (backupBreak != 0){
28       addBreakpoint(backupBreak);
29       l=backupBreak;
30       backupBreak=0;
31       resetDivisor();
32    }
33    else{
34       addBreakpoint(p);
35       l=p;
36       backupBreak=0;
37       resetDivisor();
38    }
39 }
```

Şekil 4.5 : TTTD-S Algoritması Pseudo Kodu.

TTTD-S algoritmasına TTTD algoritmasından farklı olarak yeni bir parametre eklenmiştir. Bu "switchP" parametresinin görevi, en son kesme noktası ile inceleme yapılan güncel konum arasındaki mesafe, belirlenen bu parametreden fazla olduğu durumda devreye girmektedir.

1. Ana bölen ve ikincil bölen değerlerini azaltmayı sağlar. Bunu yapabilmek için 12.satırda yer alan *switchDivisor()* fonksiyonu kullanılır.
2. Kesme noktası belirlendiğinde, ana bölen ve ikincil bölen değerleri *resetDivisor()* fonksiyonu ile birlikte başlangıç değerlerine geri döndürülür.

Yeni bir parametre eklenerek geliştirilen algoritma sayesinde bölüt boyutlarının maksimum eşiğe ulaşması engellenmiş olur, aynı zamanda çalışma süresinin kısalması ve bölüt boyutlarını ortalama bir değerde tutabilmek için sınır getirilmiş olmaktadır.

Dokümanın bu bölümüne kadar yoğun olarak veri tekilleştirmenin bölümleme kısmı ile ilgili algoritmalar üzerine yoğunlaşıldı. Verinin yedeklenmesi için iletilecek veri büyüklüğünün nasıl belirlendiği, birbirinden farklı ve zamanla geliştirilen yöntemleri ile birlikte ele alarak detaylandırmaya çalışıldı. Avantaj ve dezavantajları ile birlikte göz önüne alınarak gelişmeye açık yönleri belirlendi. Bundan sonraki bölümde ise tekilleştirme yöntemini ileri bir adıma taşıyarak bölümleme işlemlerine tamamlayıcı bir rol üstlenen senkronizasyon konusunu uygulamalı olarak inceleyeceğiz.

5. SYNCANY İLE VERİ SENKRONİZASYONU

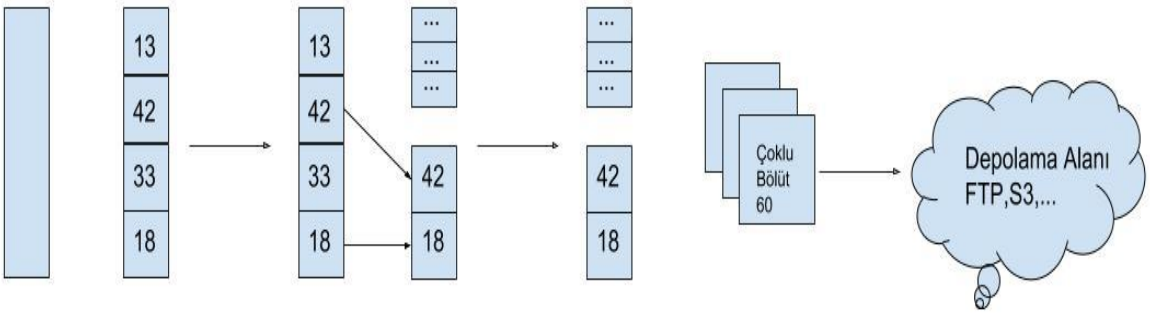
Syncany kullanıcıların herhangi bir depolama türünü kullanarak bilgisayarlarının belirli klasörlerini güvenli bir şekilde depolamalarına ve paylaşmalarına olanak tanır. Syncany açık kaynak kodludur, veri şifreleme, depolama tipi ve sağlayıcı açısından esneklik sağlar.

Dosya yedekleme ve senkronizasyon aracı olan Syncany'nin Avantajları:

1. Manüel veya otomatik olarak dosya senkronizasyonu sağlanır.
2. Herhangi bir depolama alanının kullanılmasına izin verdiğinden esneklik sağlar. Örneğin FTP, Windows dosya paylaşımı gibi.
3. Dosyalar yüklenmeden önce şifrelenir, bu sayede depolama alanına yüklenen veriler şifreli olduğu için yapılabilecek ataklara karşı daha dayanıklıdır.
4. Dosyalar istemci tarafında veri tekilleştirmesi yapılır. Uzaktan yapılan depolamalar için büyük alan tasarrufu sağlanır.
5. Dosyalar arasında versiyonlama yöntemi uygulanmaktadır, bu da eski sürümleri geri yükleme ve silinen dosyaların senkronizasyon ile güncellenmesinde kolaylık sağlar.

Senkronizasyon sağlanırken, depolama alanındaki disk kullanımı azaltmak için kullanılan veri tekilleştirme yöntemi büyük önem taşımaktadır. [12]

Syncany aracı ile kullanılan veri tekilleştirme yöntemi (Şekil 4.1)ile gösterildiği şekilde üç aşamadan oluşur:



Şekil 5.1 : Syncany aracında kullanılan veri tekilleştirme yöntemi.

1. İlk adımda dosyanın tümü indekslenerek tekil bölütler oluşturulur.
2. Veritabanı üzerinde karşılaştırmalar yapılarak yeni bölütler birleştirilerek çoklu bölüt haline getirilir. Eğer bölütler yeni ise yüklenmesi için işaretlenir. Eğer mevcut dosyalarda bulunuyor ise indeksleri yardımıyla referans gösterilir.
3. Çoklu bölütler şifrelenerek depolama alanına yüklenir.

Bu uygulamada kullanılan çoklu bölütler sayesinde, dosya başına geçen yükleme süresi ve karşılaştırmalar için yapılacak gidiş-dönüş süresi azalmaktadır.

Yukarıda bahsedilen adımlar ile veri tekilleştirmenin gerçekleştirilmesi temelde rsync algoritmasına dayanır, bu algoritmanın işleyişini ve detaylarını bir sonraki bölümde inceleyeceğiz.

5.1 Rsync Algoritması

Yaygın olarak kullanılan rsync gibi, Syncany yerel dosyaları uzaktaki bulunan kopya ile kıyaslayarak (bu en azından meta verileri ile yapılabilir) sadece değiştirilen parçaların uzaktaki depolama alanına iletilmesi sağlanır. Buradaki meta verilerden kasıt, hem dosyaların hem de dosyalarının parçalarının tarih, saat, dosya büyüklüğü ve ilgili parçalara ait sağlama toplamlarıdır.

Buradaki temel amaç, değişiklik olan kısımları büyük bloklara ayırarak her bir parça için gerekli olan indirme ve yükleme sürelerini ve bunlara karşılık gelecek olan ağ gecikmesini en aza indirmeye çalışmaktır.

Rsync algoritması adımlarını incelemek için birbirine uzaktan erişim kurulan bilgisayarları ve içerisindeki dosya örneklerini isimlendirelim. α ve β bilgisayarlarının sırasıyla D_1 ve D_2 dosyalarına erişimi olsun. D_1 ve D_2 dosyaları birbirine benzer olan eski ve yeni versiyonlar olup birbirine senkronize edilmesi istendiğinde rysnc algoritması ile aşağıdaki adımlar uygulanmaktadır: [13]

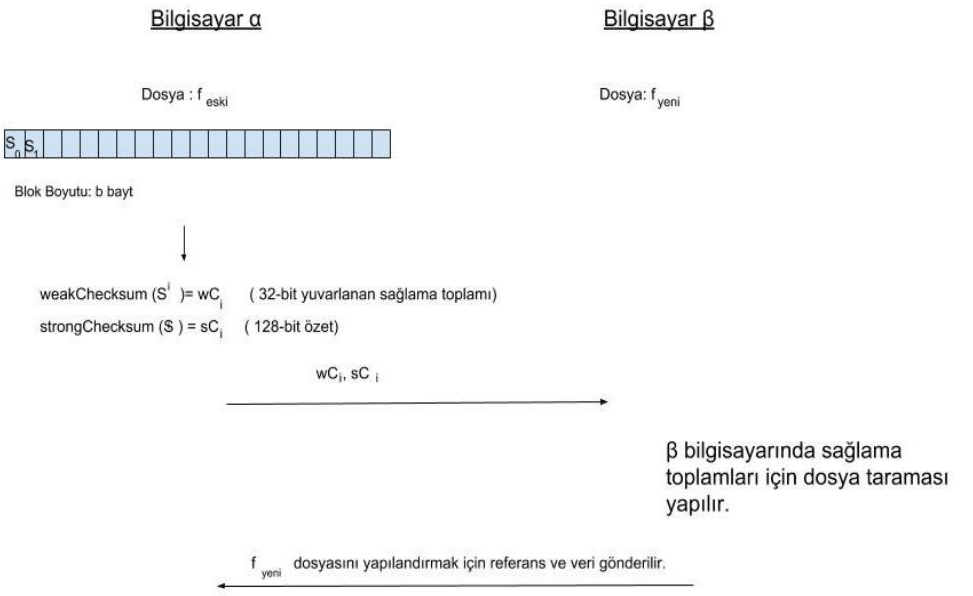
1. β bilgisayarında D_2 dosyası birbiri ile örtüşmeyen, sabit boyutlu bloklara ayrılır. Bu blokların boyutu genelde 500 ile 1000 bayt arasında seçilmektedir. Dosyanın son bloğu bu seçilen boyuttan daha az olabilmektedir.
2. Her bir blok için zayıf ve güçlü olmak üzere iki ayrı sağlama toplamı hesaplanır. Bunlar sırasıyla 32-bit ve 128 bit MD4 sağlama toplamlarıdır.
3. β bilgisayarında hesaplanan bu sağlama toplamları α bilgisayarına gönderilir.

4. α aygıtı D_1 dosyasını tarayarak , D_2 ile gelen blokların zayıf ve güçlü sağlama toplamları ile kendi depolama alanındaki sağlama toplamlarına ait değer olup olmadığı araştırılır. Bu kısım, bir sonraki bölümde değineceğimiz yuvarlanan sağlama toplamının bir özelliği ile çok hızlı bir şekilde sağlanabilir.
5. Karşılaştırmalar neticesinde, α aygıtı β aygıtına D_1 dosyasının bir kopyasını yapılandırabilmek için bir dizi bilgi gönderir. Bu bilgi D_2 dosyasının referansları veya gerçek veri olabilir.

   Gerçek verinin iletilmesi sadece, herhangi bir eşleşme sağlanmadığında yeni bir veri olduğu zaman iletilir.

   Dosyalar için referans iletilmesi ise, dosya içerisinde tarama yapıldığında aynı sağlama toplamlarının bulunması durumunda indeksleme yardımıyla bu sağlama toplamlarının gönderilmesi ile olur.

   Yukarıdaki adımların uygulanması ile rsync algoritmasını aşağıdaki şekilde özetleyebiliriz:



Şekil 5.2 : Rsync Algoritması Adımları.

5.2 Sağlama Toplamları Taraması

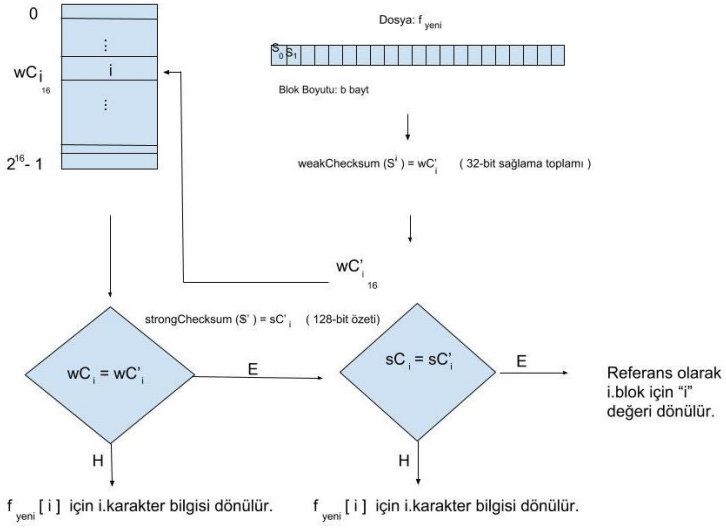
Bir önceki bölümde dosyaların bloklara ayrılması ile elde edilen sağlama toplamlarının taramadan geçmesi gerektiğinden bahsetmiştik. Bunun için öncelikle D_2 dosyasına ait bloklarının sağlama toplamları α aygıtına iletildiğinde, D_1

dosyasının herhangi bir bloğunun sağlama toplamlarına uyan değer olup olmadığı incelenir.

Temel strateji, D_1 dosyasının her bir baytı için boyutu S olan bloklar halinde 32-bit sağlama toplamı hesaplamaktır. Her bir sağlama için de bir eşleşme olup olmadığı araştırılır. Bunu uygulayabilmek için 3 ayrı seviyede inceleme yapılmaktadır.

1. İlk seviyede, 32-bit'lik sağlama toplamları için (D_2 bloklarından oluşan) 16-bit uzunluğunda özet değerler sıralanarak 2^{16} tane özet değer içeren tarama tablosu oluşturulur. Burada sıralanmış özet değerlerin bulunduğu tabloya, her bir blok için karşılık gelen değer bu indisler içerisinde var ise konumlandırılır, yok ise boş bırakılır.
2. İkinci seviye kontrolde, özet tablosunun girdileri ile sıralanmış sağlama toplamları listesinin taranması ve 32-bit sağlama toplamları ile eşleşen değerlerin aranmasını içerir. Bulunamadığı durumda bu aşamadaki kontrol sona erer.
3. Eğer ikinci seviyede eşleşme bulunur ise üçüncü seviyedeki kontrol gerçekleştirilir. Burada dosya içerisindeki ve liste üzerindeki sağlama toplamlarının 128-bit güçlü sağlama toplamları hesaplanarak kıyaslama yapılır. Eğer eşleşme sağlanırsa referans olarak indis değeri sıralanmış tablo üzerine yazılır.

Bahsedilen üç seviyede yapılan kontrolü aşağıdaki şekil ile özetleyebiliriz:



Şekil 5.3 : Sağlama Toplamı Taraması Adımları.

Bu strateji, birbirine benzer iki dosya için önemli ölçüde hesaplama ve zamandan önemli ölçüde fayda sağlar. Buna ek olarak, senkronize edilecek dosyanın yeni versiyonunun eski versiyon bloklarından oluşan bir dizi ile indekslenerek kodlanması bu faydaların sağlanması için önemli rol oynamaktadır.

6. SYNCANY SENKRONİZASYON ARACI KULLANILARAK DOSYALARI SENKONİZE ETME UYGULAMALARI

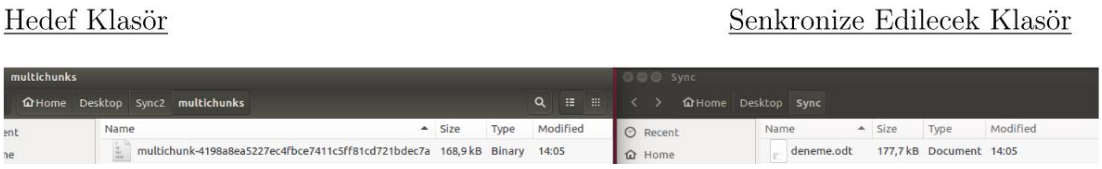
Bu bölümde, incelemiş olduğumuz senkronizasyon aracı olan Syncany ile veri tekilleştirme işlemlerini uygulamalı olarak ele alınmıştır. Birbirinden farklı uzantılı dosyalar ile örneklendirilerek bu aracın verimli olduğu ve geliştirmeye açık yönleri incelenmiştir.

Bu çalışma esnasında depolama alanı olarak kullanıcının yerel depolama alanı kullanılmış olup, herhangi bir eklenti ile çalışılmamıştır. Windows ve Linux işletim sistemine sahip bilgisayarlar ile kontroller sağlanmıştır.

6.1 LibreOffice Dokümanları ile Senkronizasyon

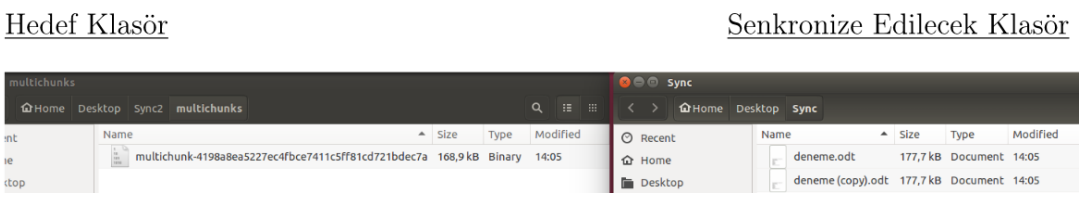
6.1.1 Libreoffice writer (.odt) dosyaları (linux)

1. Depolama alanına 177.7 kb boyutlu bir dosya eklendiğinde senkronizasyonu için 168.9 kb boyutlu çoklu bölüt (multichunk) eklediği görülmüştür (Şekil 6.1).



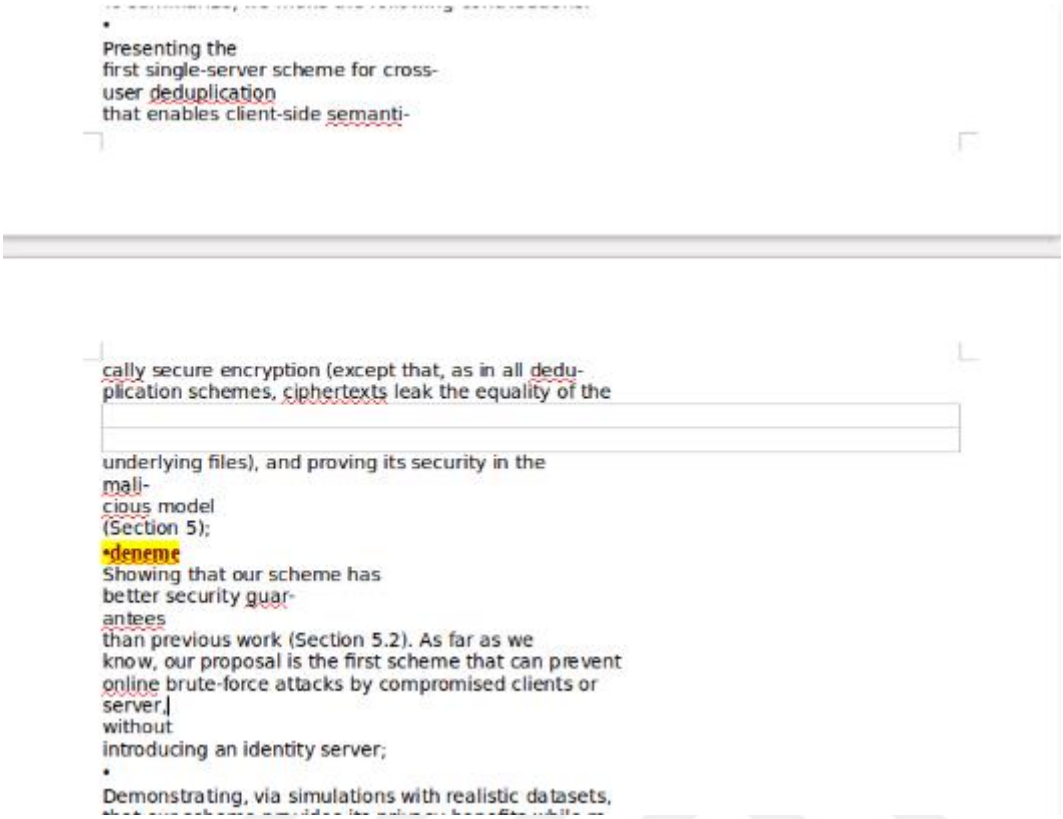
Şekil 6.1 : Yeni bir dosya eklendiğinde hedef klasör ve senkronize edilecek klasör.

2. Birebir aynı dosya eklendiği durumda, yeni bir çoklu bölüt eklenmediği görülmüştür (Şekil 6.2).

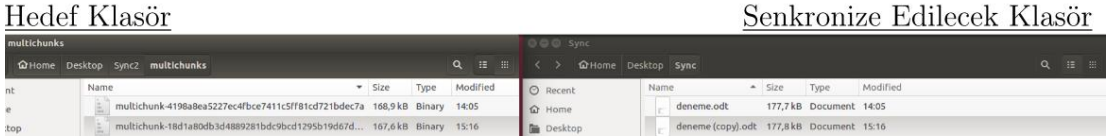


Şekil 6.2 : Dosyanın bir kopyası eklendiğinde hedef klasör ve senkronize edilecek klasör.

3. Deneme dosyası içerisine bir kelimelik ekleme yapıldığında dosya boyutu 177,8 kb olurken, 167,6 kb çoklu bölüt eklendiği görülmüştür (Şekil 6.4).

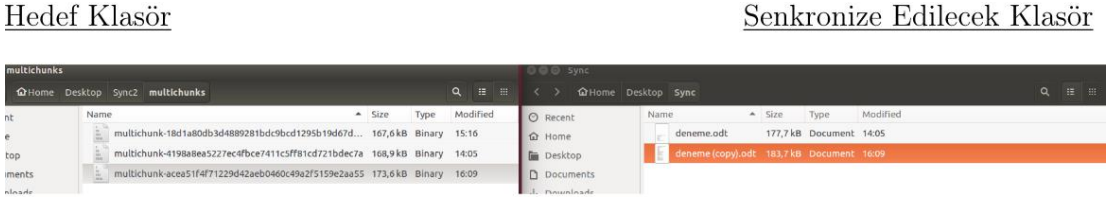


Şekil 6.3 : Deneme dosyası içerisinde ekleme yapılan kısmın ekran görüntüsü.



Şekil 6.4 : Yeni versiyonu eklenen dosya için hedef klasör ve senkronize edilecek klasör.

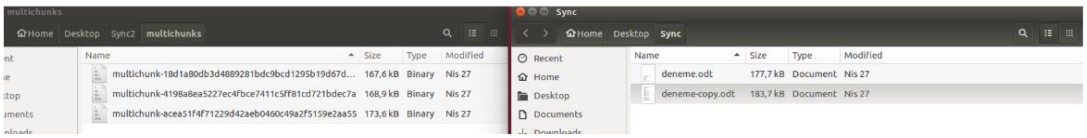
4. Deneme dosyası içerisinden veri silindiğinde; silinmeden önce eklenen çoklu bölüt boyutu 167,6 kb iken, silindikten sonra 173,6 kb olarak artış görülmüştür (silinen veri, tablo formatındadır) (Şekil 6.5).



Şekil 6.5 Dosya içerisinde veri silindikten sonra yapılan senkronizasyon işlemi.

5. Dosya ismi değiştirildiğinde herhangi bir değişiklik olmadığı gözlemlenmiştir (Şekil 6.6).

Hedef Klasör Senkronize Edilecek Klasör



Şekil 6.6 : Dosya ismi değiştirildiğinde yeni dosya için senkronizasyon işlemi.

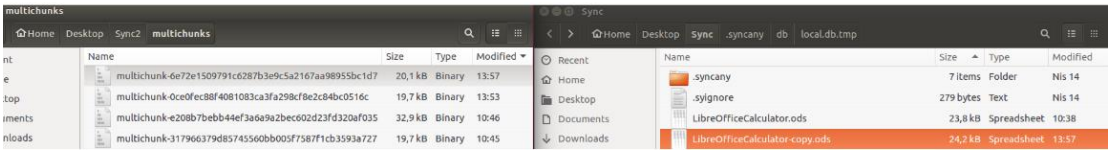
6.1.2 Libreoffice calculator (.ods) dosyaları (linux)

1. Yeni dosya eklendiğinde dosya boyutu 24.2 kb iken 20,1 kb olarak çoklu bölüt eklendiği görülmüştür (Şekil 6.7).

2. Aynı dosya farklı bir isimle eklendiğinde ise senkronize edilen klasöre herhangi bir çoklu bölüt eklenmediği görülmüştür.

3. Dosya boyutu 24.1 kb olacak şekilde içerisinden belirli bir kısım silindiğinde, 20,4 kb boyutlu yeni bir çoklu bölüt eklendiği görülmüştür.

Hedef Klasör Senkronize Edilecek Klasör

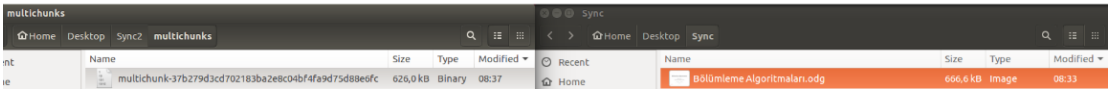


Şekil 6.7 : .ods uzantılı dosya için senkronizasyon işlemi.

6.1.3 Libreoffice impress (.odg) dosyaları (linux)

1. 666,6 kb boyutundaki dosya 626 kb boyutlu çoklu bölüt ile eklenerek senkronize edildiği görülmüştür (Şekil 6.8).

Hedef Klasör Senkronize Edilecek Klasör

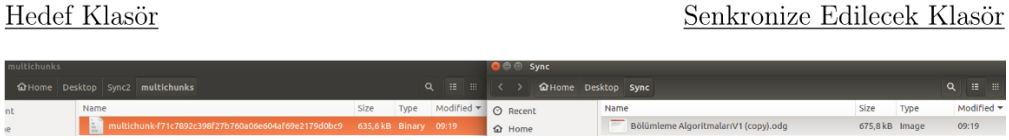


Şekil 6.8 : .odg uzantılı dosya eklendiğinde senkronizasyon işlemi.

2. Dosyada isim değişikliği yapıldığında herhangi bir değişiklik görülmemiştir.

3. Dosyanın bir kopyası eklendiğinde herhangi bir değişiklik görülmemiştir.

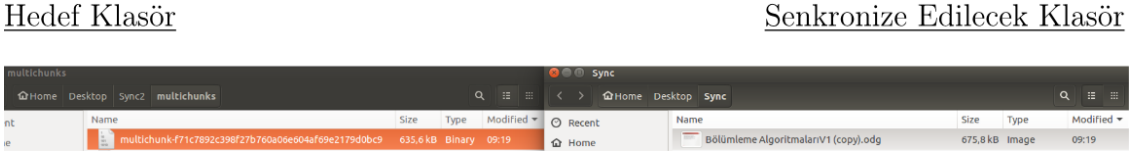
4. Aynı sunum dosyasına “deneme ekleme” metninin eklenmesiyle dosya boyutu 676,5 kb olarak artarken, 636,5 kb boyutunda çoklu bölüt eklenmiştir (Şekil 6.9).



Şekil 6.9 : Dosya güncelleme sonrasında senkronizasyon işlemi.

Şekil 6.10 : Sunum dosyası içerisine metin girişi yapılarak güncellenen kısmın ekran görüntüsü.

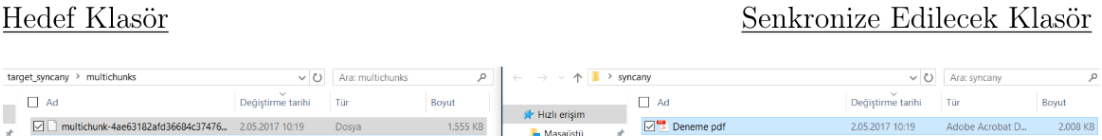
5. Aşağıdaki ekran görüntüsünde görüldüğü şekilde dosya içerisinden metin silme işlemi gerçekleştirildiğinde dosya boyutu 675,8 kb olarak azalırken, 635,6 kb çoklu bölüt eklenmiştir (Şekil 6.11).



Şekil 6.11 : Dosya içerisinden metin silme işlemi gerçekleştiğinde senkronizasyon işlemi.

6.2 PDF dokümanları ile senkronizasyon (windows)

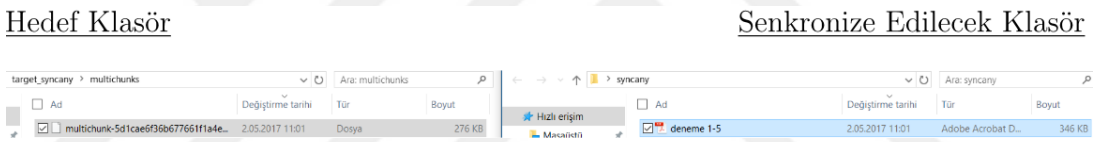
1. 2008 kb boyutundaki yeni bir pdf dosyası eklendiğinde 1555 kb boyutunda çoklu bölüt eklendiği görülmüştür (Şekil 6.12).



Şekil 6.12 : Yeni bir pdf dosyası eklendiğinde senkronizasyon işlemi.

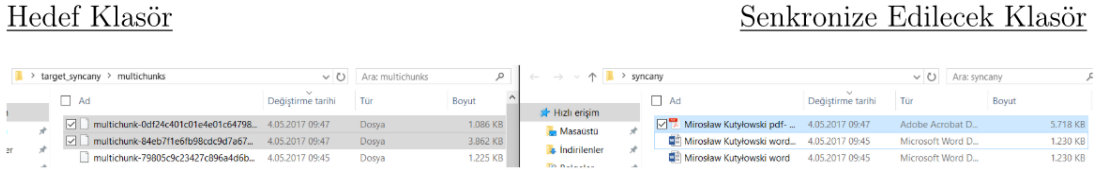
2. Aynı dosyanın kopyaları eklendiğinde herhangi bir çoklu bölüt eklenmediği görülmüştür.

3. Dosyanın ilk 5 sayfası için "deneme 1-5" isimli dosya eklendiğinde, 276 kb boyutunda çoklu bölüt eklenmiştir (Şekil 6.13).



Şekil 6.13 : Dosyanın belli bir kısmının tekrar eklenmesi senkronizasyon işlemi.

4. Bir dosyanın birebir aynı olan word formatı ile pdf formatı kaydedildiğinde her biri için ayrı çoklu bölütlerin oluşturulduğu gözlemlenmiştir (Şekil 6.14).



Şekil 6.14 : Farklı formatta aynı dosyanın senkronizasyon işlemi.

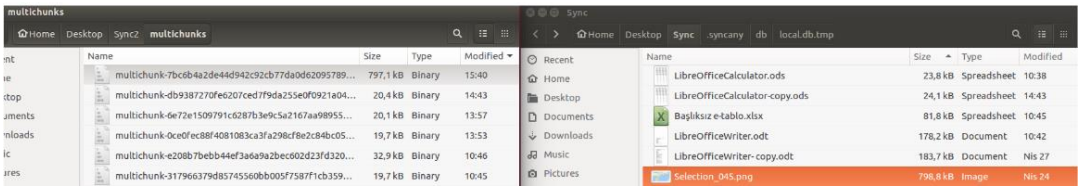
6.3 Görüntü Dosyaları ile Senkronizasyon

6.3.1 Ubuntu linux işletim sistemi ile uygulaması

1. Dosya boyutu 798,8 kb iken, senkronize edilen klasörde 797,1 kb olacak şekilde çoklu bölüt oluşturulmuştur (Şekil 6.15).

2. Aynı görüntünün bir kopyası eklendiğinde, herhangi bir çoklu bölüt eklenmemiştir.

Hedef Klasör Senkronize Edilecek Klasör

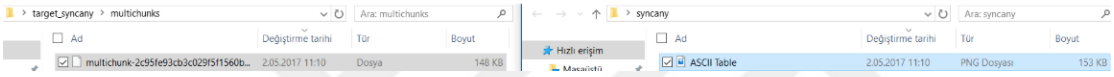


Şekil 6.15 : Görsel (image) dosyası eklendiğinde senkronizasyon işlemi (Linux).

6.3.2 Microsoft windows işletim sistemi ile uygulaması

1. Ekran görüntüsü boyutu 153 kb iken, 146 kb boyutunda çoklu bölüt eklenmiştir (Şekil 6.16).

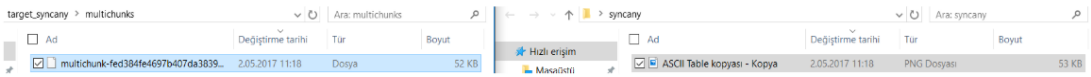
Hedef Klasör Senkronize Edilecek Klasör



Şekil 6.16 : Görsel (image) dosyası eklendiğinde senkronizasyon işlemi (Windows).

2. Bir ekran görüntüsünün 53 kb boyutundaki bir kısmı kesilip yeni bir dosya olarak kaydedildiğinde 52 kb boyutunda çoklu bölüt eklendiği görülmüştür (Şekil 6.17).

Hedef Klasör Senkronize Edilecek Klasör



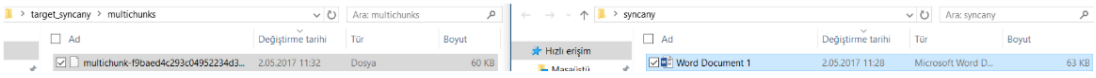
Şekil 6.17 : Dosyanın belli bir kısmının yeni dosya olarak eklenmesi ile senkronizasyon işlemi.

6.4 Microsoft Office Dokümanları ile Senkronizasyon

6.4.1 Microsoft word (.docx) dosyaları (windows)

1. 63 kb boyutunda bir word dosyası eklendiğinde 60 kb boyutunda yeni bir çoklu bölüt eklenmiştir (Şekil 6.18).

Hedef Klasör Senkronize Edilecek Klasör

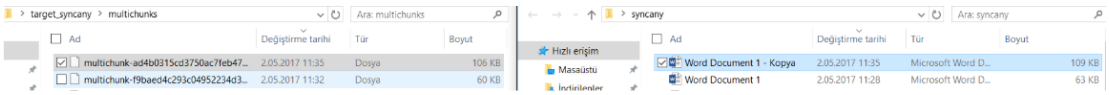


Şekil 6.18 : Yeni word dosyası eklendiğinde senkronizasyon işlemi.

2. Dosyanın kopyası eklendiğinde herhangi bir değişiklik olmadığı görülmüştür.

3. Dosya içerisinde bulunan veriler kopyalanarak dosya içerisine eklendiğinde 109 kb boyutunda bir dosya oluşturulmuş, 106 kb boyutunda çoklu bölüt eklenmiştir (Şekil 6.19).

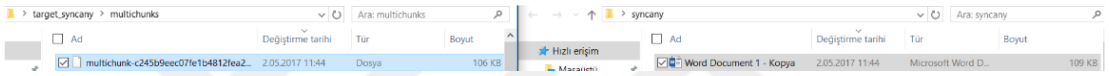
Hedef Klasör Senkronize Edilecek Klasör



Şekil 6.19 : Dosya içerisine benzer verilerin bulunduğu kopyalar eklendiğinde senkronizasyon işlemi.

4. Dosya içerisinden bir kısım silinip kaydedildiğinde dosya boyutu kadar çoklu bölüt eklendiği görülmüştür (Şekil 6.20).

Hedef Klasör Senkronize Edilecek Klasör



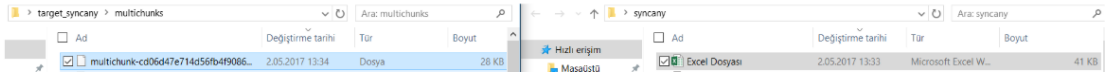
Şekil 6.20 : Dosyadan veri silindiğinde yeni dosyanın senkronizasyon işlemi.

6.4.2 Microsoft excel (.xlsx) dosyaları (windows)

1. 41 kb boyutlu bir excel dosyası eklendiğinde 28 kb boyutlu bir çoklu bölüt

Eklenmiştir (Şekil 6.21).

Hedef Klasör Senkronize Edilecek Klasör

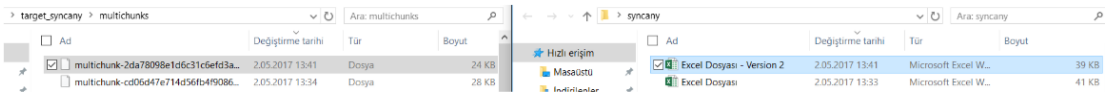


Şekil 6.21 : Yeni bir Excel dosyasının eklenmesiyle senkronizasyon işlemi.

2. Dosyanın kopyası eklendiğinde herhangi bir değişiklik olmamıştır.

3. Dosya içerisinden bir kısım silindiğinde dosya boyutu 39 kb'a düşerken, 24 kb boyutunda çoklu bölüt eklendiği görülmüştür (Şekil 6.22).

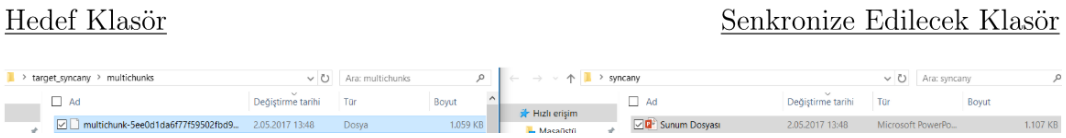
Hedef Klasör Senkronize Edilecek Klasör



Şekil 6.22 : Dosya içerisinde silme işlemi yapıldıktan sonra senkronizasyon işlemi.

6.4.3 Microsoft powerpoint (.pptx) dosyaları (windows)

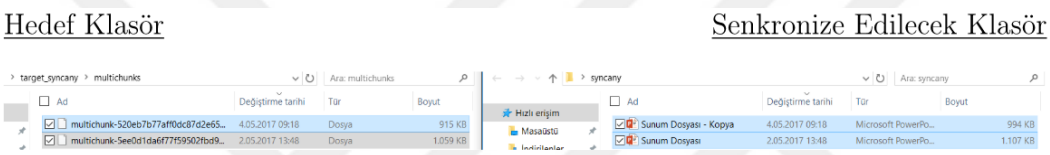
1. 1017 kb boyutunda bir sunum dosyası eklendiğinde 1059 kb boyutunda çoklu bölüt eklenmiştir (Şekil 6.23).



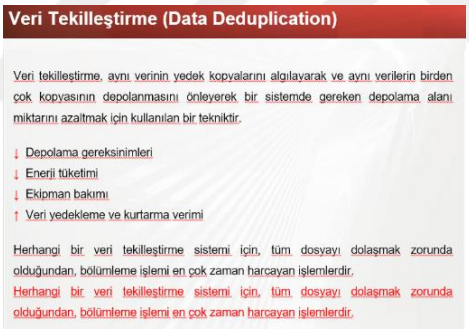
Şekil 6.23 : Powerpoint dosyası eklendiğinde senkronizasyon işlemi.

2. Aynı dosyanın kopyası eklendiğinde herhangi bir çoklu bölüt eklenmemiştir (Şekil 6.24).

3. Dosya içerisinde bir cümle kopyalanarak tekrar eklendiğinde 915 kb boyutunda yeni bir çoklu bölüt eklenmiştir (Şekil 6.25).

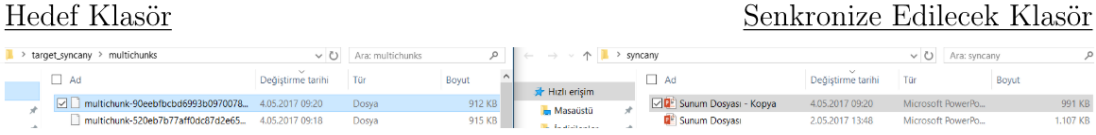


Şekil 6.24 : Dosya kopyasına ekleme yapıldığında senkronizasyon işlemi.



Şekil 6.25 : Sunum dosyası içerisine metin ile ekleme yapılan kısmın ekran görüntüsü.

4. Sunum dosyası içerisinden bir sayfa silindiğinde benzer şekilde yeni bir 912 kb boyutunda çoklu bölüt eklendiği görülmüştür (Şekil 6.26).



Şekil 6.26 : Sunum dosyası içinden sayfa silindiğinde senkronizasyon işlemi

6.5 Text Dosyaları ile Senkronizasyon (Windows)

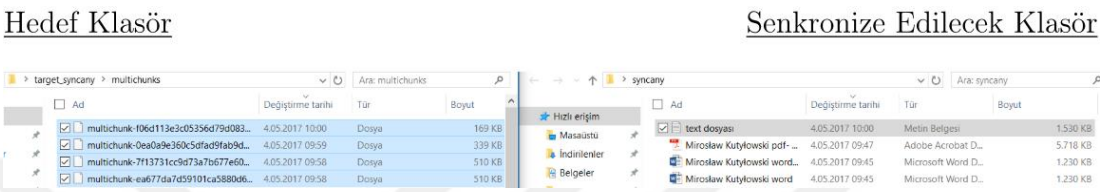
Bir text dosyası üzerinde küçük değişiklikler yapıldığında çoklu bölüt boyutlarının

azaldığı gözlemlenmiştir.

Yapılan işlemler ile sırasıyla aşağıda verilmiştir:

- ‘Deneme’ kelimesi dosyanın ortasına eklenmiştir, sonucunda 510 kb yerine 339 kb boyutunda çoklu bölüt eklenmiştir (Şekil 6.27).
- Dosya içerisinden bir paragraf (2 kb) alınarak aynı dosyanın sonuna eklenmiştir,

sonucunda 510 kb yerine 169 kb boyutunda çoklu bölüt eklenmiştir.



Şekil 6.27 : Text dosyası ile senkronizasyon işlemi.

7. SONUÇ

Syncany'nin çalışma prensibine göre her yeni dosya geldiğinde; dosya önce bölütlerine ayrılır, bu bölütlerin daha önceden var olup olmadığına bakılır, eğer yoksa çoklu bölüt olarak paketlenir. Daha sonra çoklu bölütler sıkıştırılır ve depolama alanında saklanır.

Uygulamalı olarak incelenen örnekler neticesinde aşağıdaki sonuçlar elde edilmiştir:

1. Yeni eklenen dosyalar sıkıştırıldığı için kendi boyutlarından daha küçük boyutta çoklu bölütleri oluşturmaktadır.
2. Text dosyaları için güncelleme yapıldığında, yeni dosyanın senkronizasyonu sağlanırken çok daha küçük boyutlu çoklu bölütlerin eklendiği gözlemlenmiştir.
3. Microsoft Office ve Libre Office Dosyaları için aynı dosya üzerinde yapılan ekleme veya silme işlemlerinde çok küçük değişiklikler yapılmasına rağmen tüm dosya boyutuna yakın çoklu bölütlerin eklendiği görülmüştür.

   Sycany aracı ile Microsoft Office ve Libre Office dosyalarına nazaran text dosyalarında senkronizasyon sağlanması daha uygundur.
4. Dosyaların birebir aynı olan kopyaları eklendiğinde senkronizasyon için herhangi bir çoklu bölütün eklenmediği görülmüştür.
5. Dosya içeriğinde değişiklik yapılmaksızın isim değişikliği yapıldığında senkronizasyon için herhangi bir çoklu bölüt eklenmediği görülmüştür.
6. Bir dosyanın farklı formatlardaki (word-pdf) versiyonları arasında herhangi bir tekilleştirme yönteminin uygulanmadığı görülmüştür.
7. Dosya içerisinden belirli bir kısım alınarak farklı bir dosya oluşturulup eklendiğinde, tekilleştirme işleminin yapılmadığı görülmüştür. Birebir aynı veri başka bir çoklu bölütte bulunmasına rağmen yeni çoklu bölütlerin eklendiği gözlemlenmiştir.
8. Bir dosya içerisinde birden fazla aynı veri bulunduğunda dosya bazında veri tekilleştirmenin yapılmadığı gözlemlenmiştir.

Elde edilen sonuçlar incelendiğinde, ağ bandı genişliğini düşürmek ve disk alanının gereksiz kullanımını önlemek için çoklu bölütlerin kullanılması hedeflenen veri tekilleştirme yöntemlerinin gerçeklenebilmesi için önemli rol oynamaktadır. Birebir aynı dosyaların uzaktaki veri depolama alanına iletilmesinin sağlanması için sadece indeks iletilmesinin sağlanması yeterli olacağından, veri alışverişi minimum seviyede tutulmaktadır. Bunların yanında dosya üzerinde yapılan değişikliklerin senkronizasyonunda tekilleştirmeye duyarlılığının arttırılması için Syncany üzerinde iyileştirmeler yapılması ile daha da verimli sonuç alınmasını sağlayacaktır.

KAYNAKLAR

[1] **A. Muthitacharoen, B. Chen, D. Mazieres**, (2001). A Low-Bandwidth Network File System, *Proc. 18th ACM Symp. Operating Systems Principles*.

[2] **M. S. Kiraz**, (2016). Solving the Secure Storage Dilemma: An Efficient Scheme for Secure Deduplication with Privacy-Preserving Public Auditing, *IIACR Cryptology ePrint Archive 2016.*

[3] **Moh, Chang**, (2010) A Running Time Improvement for the Two Thresholds Two Divisors Algorithm, *ACM SE '10 Proceedings of the 48th Annual Southeast Regional Conference 2010*

[4] **Mandagere, N., Zhou, P., Smith, M.A., Uttamchandani, S.**, (2008). Demystifying data deduplication, *ACM, New York.*

[5] **Sean Quinlan and Sean Dorward**, (2002). Proceedings of the FAST '02 Conference on File and Storage Technologies, *Doubletree Hotel, Monterey, California, USA.*

[6] **J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W Weimer, C. Wells, and B. Zhao.**, (2000). Oceanstore: An architecture for globalscale persistent store, *In Proc. ASPLOS'2000, Cambridge, MA.*

[7] **Min, Jaehong, Daeyoung Yoon, and Youjip Won.**, (2011). Efficient deduplication techniques for modern backup operation, *Computers, IEEE Transactions.*

[8] **Y. Won, R. Kim, J. Ban, J. Hur, S. Oh, J. Lee**, (2008). Prun: Eliminating Information Redundancy for Large Scale Data Backup System, *Proc. IEEE Int'l Conf. Computational Sciences and Its Applications(ICCSA '08).*

[9] **Daehee Kim; Sejun Song; Baek-Young Choi**, (2017). Data Deduplication for Data Optimization for Storage and Network Systems, *eBook:Document.*

[10] **Broder, A. Z.** (1993). Some applications of Rabin's fingerprinting method. *In Sequences II (pp. 143-152). Springer, New York, NY.*

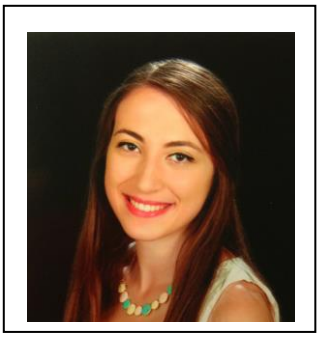
[11] **Eshghi, K., & Tang, H. K**. (2005). A framework for analyzing and improving content-based chunking algorithms. *Hewlett-Packard Labs Technical Report TR, 30.*

[12] **Philipp C. Heckel**, (2013). Minimizing remote storage and synchronization

time using deduplication and multichunking. Syncany as an example, *blog.philippheckel.com*

[13] **Tridgell, A., & Mackerras, P.**, (1996). The rsync algorithm.

ÖZGEÇMİŞ



Ad-Soyad : Duygu COŞGUN

Doğum Tarihi ve Yeri : 20.08.1991 / İstanbul

E-posta : duygu-cosgun@windowslive.com

ÖĞRENİM DURUMU:

- **Lisans** : 2013, Mimar Sinan Güzel Sanatlar Üniversitesi, Fen Edebiyat Fakültesi, Matematik
- **Yükseklisans** : 2018, İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Matematik Mühendisliği

MESLEKİ DENEYİM VE ÖDÜLLER:

- 2009-2013 yılları arasında okuduğu Mimar Sinan Güzel Sanatlar Üniversitesi'nden bölüm üçüncülüğü ile mezun olmuştur.
- 2012-2013 güz döneminde öğrenci değişim programı ile İtalya'da öğrenim hakkı kazanmış ve Politecnico di Milano üniversitesinde Matematik Mühendisliği üzerine eğitim görmüştür.
- 2012 yılında İtalya - Politecnico di Milano Üniversitesi'nde italyanca dil eğitimi alarak başarılı bir şekilde tamamlamıştır.
- 2013 yılında İngiltere - Eastbourne School of English dil okulunda dil eğitimi alarak başarılı bir şekilde tamamlamıştır.

- 2013-2014 Marmara Üniversitesi-Eğitim Bilimleri Fakültesi'nde Pedagojik Formasyon eğitimini tamamlayarak sertifika almaya hak kazanmıştır.
- 2014 yılında İTÜ24 şirketinden Yazılım Uzmanlığı Eğitimi alarak başarı sertifikası almaya hak kazanmıştır.
- 2014-2015 yılları arasında Vodafone Türkiye şirketinde Bilişim Teknolojileri departmanında yarı zamanlı çalışan olarak görev almıştır.
- 2015-2016 yılları arasında Yapı Kredi Bankacılık Üssü yerleşkesi Bilişim Teknolojileri departmanında İş Analisti olarak görev almıştır.
- 2016 yılında SDLC (Software Development Life Cycle) eğitimi alarak sertifika almaya hak kazanmıştır.
- 2016 yılında IIBA CBAP Uluslar Arası İş Analisti Eğitimi (BA Works) alarak sertifika almaya hak kazanmıştır.
- 2017 yılında Tübitak'ta Ar-Ge Mühendisi olarak görev almıştır.
- 2017-2018 güz döneminde İstanbul Şehir Üniversitesi Bilgi Güvenliği Mühendisliği yüksek lisans programında özel öğrenci olarak ders almıştır.
- Haziran 2017'den itibaren OBSS Bilişim (Open Business Software Solutions) şirketinde Uzman İş Analisti olarak görev almaktadır.