İLLER BANKASI ANONİM ŞİRKETİ

İL-CAS PROJESİ İÇİN BÜYÜK VERİ PLATFORMUNUN SEÇİMİ VE PROJEYE KATACAĞI PERFORMANS

Hatice Tül Kübra BAMYACI

UZMANLIK TEZİ

NİSAN 2017

İLBANK

TÜRKİYE'NİN YAPICI GÜCÜ

İLLER BANKASI ANONİM ŞİRKETİ

İL-CAS PROJESİ İÇİN BÜYÜK VERİ PLATFORMUNUN SEÇİMİ VE PROJEYE KATACAĞI PERFORMANS

Hatice Tül Kübra BAMYACI

UZMANLIK TEZİ

Tez Danışmanı (Kurum)
Ercan ÖZBAY

Tez Danışmanı (Üniversite)
Doç. Dr. Hasan Şakir BİLGE

ETİK BEYAN

İLLER BANKASI ANONİM ŞİRKETİ Uzmanlık Tezi Yazım Kurallarına uygun olarak hazırladığım bu tez çalışmasında; tez içinde sunduğum verileri, bilgileri ve dokümanları akademik ve etik kurallar çerçevesinde elde ettiğimi, tüm bilgi, belge, değerlendirme ve sonuçları bilimsel etik ve ahlak kurallarına uygun olarak sunduğumu, tez çalışmasında yararlandığım eserlerin tümüne uygun atıfta bulunarak kaynak gösterdiğimi, kullanılan verilerde herhangi bir değişiklik yapmadığımı, bu tezde sunduğum çalışmanın özgün olduğunu, bildirir, aksi bir durumda aleyhime doğabilecek tüm hak kayıplarını kabullendiğimi beyan ederim.

Hatice Tül Kübra BAMYACI

04/ 04/ 2017

İL-CAS Projesinde Büyük Veri Platformunun Seçimi ve Projeye Katacağı Performans
(İLBANK Uzmanlık Tezi)

Hatice Tül Kübra BAMYACI

İLLER BANKASI ANONİM ŞİRKETİ

Nisan 2017

ÖZET

Teknolojide yaşanan gelişmeler ve bilgisayarların insan hayatının her safhasında yer alması gibi nedenlerle üretilen veri miktarı artmaktadır. Verilerin kullanıldığı uygulama çeşitliliği arttıkça; veri çeşitliliği de artmakta, daha karmaşık veri tipleri oluşmaktadır. Geleneksel veritabanı yönetim sistemlerinin oluşan büyük miktarlardaki ve karmaşık yapılardaki verileri saklama, yönetme ve işleme konusunda yetersiz kalmasıyla, büyük veri kavramı ortaya çıkmıştır. Geleneksel veritabanlarının yönetemediği verileri, saklamak ve işlemek için yatay olarak ölçeklendirilebilen NoSQL veritabanları kullanılmaktadır. Tek bir veri aracının yetersiz kaldığı daha karmaşık sistemlerde büyük miktardaki verileri anlık olarak işleyebilmek için Lambda mimarisi ortaya çıkmıştır. Geleneksel veri çözümlerinde yaşanan sorunlar, büyük veri sistemlerinde olması istenilen özellikler, Lambda mimarisi ve İL-CAS projesinde Lambda mimarisinin uygulanabilirliği araştırılmış; bu mimaride kullanılabilecek depolama araçları önerilmiştir.

Anahtar Kelimeler	:	Büyük veri, İL-CAS, lambda mimarisi, geleneksel veritabanları
Sayfa Adedi	:	72
Tez Danışmanı (Kurum)	:	Ercan ÖZBAY
Tez Danışmanı (Üniversite)	:	Doç. Dr. Hasan Şakir BİLGE

Choosing Big Data Tools for ILCAS and The Performance They Provide the Project

İL-CAS Projesinde Büyük Veri Platformunun Seçimi ve Projeye Katacağı Performans

(ILBANK Expertise Thesis)

Hatice Tül Kübra BAMYACI

ILLER BANKASI ANONİM ŞİRKETİ

April 2017

ABSTRACT

The size of generated data is increasing because of the reasons like developments in technology and that computers are used in evert aspect of people's lives. The more range of software applications using data increases, complicated data types occurs. It is called as big data which are data sets came from different resources. Traditional database management systems failed to manage this data sets. NoSQL database systems are used to manage and store the data which traditional database systems cannot administer. Lambda architecture is explored to manage big real time data where the systems that using a single database tool is insufficient. The problems which are seen in traditional data solutions, the features which are desired in big data systems are discussed, Lambda architecture and applicability of Lambda architecture in ILCAS are searched and storage tools which can be used in the architecure are suggested.

Key Words	:	Big data, IL-CAS, lambda architecture, traditional databases
Page Numbers	:	72
Supervisor (Institution)	:	Ercan ÖZBAY
Supervisor (University)	:	Assoc. Prof. Hasan Şakir BİLGE

TEŞEKKÜR

Çalışmam esnasında verdikleri destekten ötürü tez danışmanlarım Ercan ÖZBAY ve Doç. Dr. Hasan Şakir BİLGE'ye, müdürüm Ali Rıza DEMİREL'e, iş arkadaşlarımdan Hüseyin ELLEZER'e ve başta eşim Ömer Ersan BAMYACI olmak üzere tüm aileme teşekkür ederim.

İÇİNDEKİLER

Sayfa

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

Şekil **Sayfa**

RESİMLERİN LİSTESİ

Resim **Sayfa**

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılan kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Kısaltmalar	**Açıklamalar**
Banka	İlbank A.Ş.
İL-CAS	İlbank Coğrafi Arşiv Sistemi
OGC	Open Geospatial Concortium
WFS	Web Feature Service
WMS	Web Map Service

GİRİŞ

Son 10 yılda, teknolojide olan değişimler, internetin yaygınlaşması gibi birçok nedenle, üretilen verinin miktarı hızlı bir artış göstermiştir. Her saniye 30 gigabytetan daha fazla üretilmekte ve veri üretim hızı da her geçen gün artmaktadır. Veri miktarı artıkça birçok kurum verileri yönetmek konusunda zorlanmaktadır. Birçok kurum gigabaytlar hatta terabaytlarca veri saklamaktadır. Tablolarda saklanan biçimli veri tipleri gibi, coğrafi veriler, pdf ya da word dosyaları, videolar gibi biçimlendirilmemiş veri tiplerinin de olması veri sistemlerinin karmaşasını artırmaktadır.

Verinin bu kadar hızlı bir şekilde büyüyor olması, çalışan sistemleri de kaçınılmaz bir şekilde etkilemektedir. Geleneksel veri tabanlarının sınırlarını zorlaması ve büyük miktardaki verileri ölçeklendirmekte başarısız olmaları gibi birçok nedenden dolayı büyük veri kavramı ortaya çıkmıştır.

Büyük veri kavramı altında irdelenebilecek doğası gereği yoğun olan verilerin başında coğrafi veriler gelmektedir. Coğrafi büyük veri kavramı genellikle mevcut hesaplama sistemlerinin kapasitesini aşan coğrafi veri setleri olarak adlandırılır.

İL-CAS, Bankamıza ait coğrafi verilerin OGC standartlarında saklanmasını, sunulmasını ve servis edilmesini sağlayan bir coğrafi arşiv sistemidir. İL-CAS, Bankamızın kurulduğu yıldan itibaren üretilen coğrafi verilerin sistemik olarak saklanabilmesini sağlayan, böylelikle ülkemizin kentsel gelişim aşamalarının gözlemlenebileceği sayısal bir ortamdır. İL-CAS, coğrafi verilerin yapısından kaynaklanan veri yoğunluğundan dolayı test ortamında istenilen performansta çalışamamıştır.

Bu tez kapsamında, geleneksel veri çözümlerinin büyük verileri yönetirken yaşadığı sorunlar, bu sorunlara çözüm olarak sunulan Lambda mimarisi, bu mimarinin İL-CAS'a uygulanabilirliği ve sağlayabileceği performans artırımı araştırılmıştır.

1. BÜYÜK VERİ

Büyük veri geleneksel veri çözümlerinin yönetmekte başarısız kaldığı veri setlerine verilen addır. Büyük veri sistemlerini tanımlayan ve "5V" olarak bilinen dört anahtar kavram aşağıda gösterilmektedir:

- Volume (Hacim): Verinin miktarını ifade eder. Her an her saniye email, fotoğraf, mesaj, video klip gibi büyük miktarlarda veri üretilmektedir. Kendi Bankamız veri büyüklüğünü düşündüğümüzde, Bankamızın 80 yılı aşkın arşiv verisi bulunmaktadır. İL-CAS Projesi kapsamında tüm arşiv verileri sisteme aktarılamadığı durumda bile, proje kapsamında taranan ve indekslenen paftaların, aktarılan sayısal verilerin büyüklüğü, taranan paftalara ve sayısal projelere ait verilerin büyüklüğü yaklaşık olarak 20 TB kadardır.

- Variety (Çeşitlilik): Verinin çeşitliliğini ifade eder. Geçmişte veri denildiğinde akla ilişkisel veri tabanlarında tablolarda tutulan yapısal veriler akla gelirdi. Günümüzde her veri kaynağı farklı tipte veriler üretmektedir. Üretilen fotoğraf, video, coğrafi veriler gibi tanımlı (unstructured)olmayan birçok veri tipi bulunur. Oluşturulan sistemler, hem tanımlı (structured) hem de yapısal olmayan verilerin birbirini anlayacağı nitelikte olmalıdırlar. Bankamız projesi İL-CAS'da, coğrafi veriler, taranmış ve koordinatlandırılmış resimler, projelere ait zaman, rakamsal değerler gibi birçok veri tipi mevcuttur. Projede veri çeşitliliği oldukça fazladır.

- Veracity (Doğruluk): Verinin doğruluğunu ve güvenirliğini ifade eder. Veriler doğru katmandan doğru kişilere ulaştırılacak güvenliğe sahip olan sistemlerde saklanmalıdır. Veriler ile ilişki olmayan kişiler verilere erişememelidir.

- Velocity (Hız): Yeni gelen verinin büyüklüğünü ifade eder. Günümüzde her an her saniye e-mail, fotoğraf, mesaj, video klip gibi büyük miktarlarda veri üretilmektedir. Üretilen veri miktarı arttığı için, bu veriyi kullanacak işlem sayısının ve çeşitliliğinin de aynı hızda artmasına neden olur.

- Value (Değer): Veriden değer oluşturulması anlamındadır. Büyük veri sistemlerine sahip olan kurumlar, bu sistemden faydalanmalı, anlık verilen kararlarda bu sistemlerden faydalanmalıdırlar.

1.1. Geleneksel Veritabanlarında Yaşanan Sorunlar

1.1.1. Ölçeklendirme

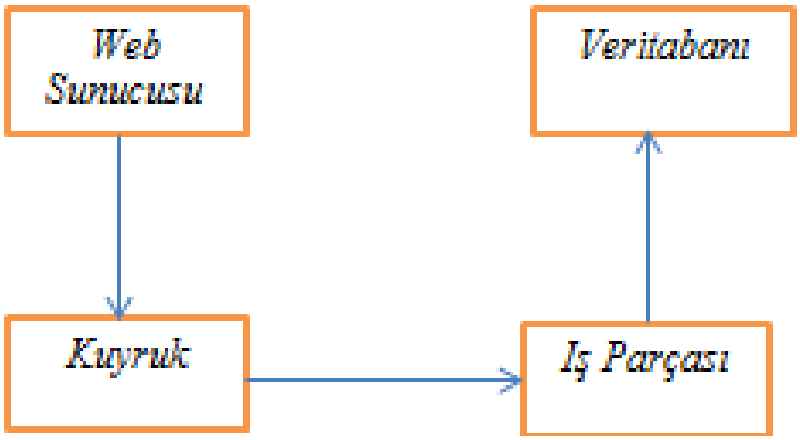
Geleneksel veri veritabanlarının yaşadığı en önemli sorunların başında; veri setinin veritabanın yanıt veremeyeceği kadar büyümesi ve veritabanın veri yükünü kaldıramaması gelmektedir. Bu konuyu daha iyi anlamak için şöyle bir web uygulamasının örneklemesi yapılabilir: Verilen örnekte uygulama kullanıcıların görmek istedikleri sayfaların görüntülenme sayısını takip etmektedir.

Çizelge 1.1. Örnek uygulamanın tablo şeması

Sütun Adı	*Tipi*
id	integer
kullanici_id	integer
url	varchar(255)
goruntuleme_sayisi	bigint

Çizelge 1.1'de görüldüğü üzere; uygulama, veritabanında üzerinde siteye giren kullanıcıların IP adreslerini ve o IP adreslerine ait sayfa görüntüleme sayısını tutmaktadır. Yeni bir sayfa görüntülendiğinde, kullanıcının web sayfası uygulamanın web sunucusuna ping atmakta, uygulama da veritabanına görüntülenen sayfanın linki, görüntülenme sayısını ve kullanıcının kimlik numarasını kaydetmektedir.

Uygulama büyüyüp kullanıcı sayısı artış gösterdiğinde, sistemin yükü kaldıramaz hale gelebileceği öngörülmektedir. Bu durumda veritabanı kayıt işlemi için zaman aşımı hatası verebilir.



Şekil 1.1 Kuyruk ve işlem parçası ile komut listesi

Şekil 1.1'deki döngüde sistemin aynı anda kaldıramayacağı kadar talep olduğunda, zaman aşımı hatası olmaması için, veritabanı ile web sunucusu arasına 100 adet talebin eklenebildiği bir kuyruk eklenmiştir. Yeni bir sayfa görüntülendiğinde, bu istek kuyruğa eklenmektedir. Kuyruktaki talep sayısı 100'ü bulduğunda iş parçası kuyruktan aldığı verileri veritabanına yazma işlemini gerçekleştirmektedir.

Uygulama daha çok kullanılmaya başlandığında, veritabanı yeniden aşırı yüklenecektir. İş parçası sayısı artırılıp, veritabanı işleri paralel yaptırılmak istenebilir ama bir süre sonra bu da yetersiz kalacaktır. [1]

İş parçaları yetersiz kaldığında, veritabanı yatay parçalama ile birden fazla parçaya dağıtılmaktadır. Bu yöntem, kullanılan en yaygın yöntemdir. Bu yöntemde; her bir parça, hash fonksiyonu yardımıyla bir anahtar ile ilişkilendirilmektedir. Bu işlem bittiğinde, uygulama kodunun her anahtar için hangi parçayı bulacağını bilmesi gerekmektedir. Bu yüzden konfigürasyon dosyasına parça sayısı eklenip, uygulama yeniden başlatılmaktadır. [1,2]

Uygulama daha da çok kullanılmaya başlandığında, yük ile başa çıkabilmek için veritabanı daha fazla parçaya ayrılmaktadır. Her seferinde bu sürecin daha zor olması muhtemeldir. Çünkü koordine edilmesi gereken birçok iş vardır. Bu sureci kolaylaştırmak için veritabanı üzerine yeniden parçalamayı yapacak tek bir komut dizi yazılabilir. Ama bu işlem de aşırı yavaş olacaktır. Yeniden parçalamayı ve aktif olan birçok komut dizinini paralel olarak yapmak gerekir ki; bu da çok zor bir süreçtir. Yeniden parçalama işlemini

yapıp, uygulama güncellenmemesi birçok kayıtın yanlış parçalara kaydedilmesine neden olabilir. Böyle bir senaryo gerçekleştiğinde parçalardaki yanlış verileri doğru parçalara taşımak gerekir.

1.1.2. Hata toleransı

Parçalama yapılan veritabanında sunuculardan biri çöktüğü zaman, o sunucuya ait verilere erişim sağlanamayacaktır. Bu durumda aşağıdaki yollar izlenebilir:

1. Sistem ulaşılamayan veritabanı parçaları için parçaların ulaşılamadığı süreçte yeni gelen talepleri ek bir kuyruğa atacak şekilde dizayn edilebilir. Sistem tekrar çalıştırılıp parça ulaşılır olduğunda ek kuyrukta bulunan talepler tekrar parçanın kuyruğuna eklenir.
2. Veritabanı kapasitesinin bir kısmını kullanarak her parçaya bir köle eklenebilir. Böylece efendi aşağı indiği zaman, her efendinin yedeği olduğundan kullanıcılar en azından verileri görebilecektir.

Bu çözümler sorunları kısmen çözebilir ama bu işlemler veritabanı yöneticileri için ek yük demektir.

1.1.3. Karışıklık

Bölüm 1.1.1'de yapılan örneklemede basit bir web uygulaması için sistemin karmaşasının nasıl arttığı görülmektedir. Kuyruklar, parçalama işlemleri, parçalanma için yazılan kod dizileri gibi birçok etken ile uygulamalar karışık hale gelmektedir. Uygulama geliştirmek için yalnızca veritabanı şemasını bilmek yeterli değildir. Kodun hangi parça ile konuşacağını bilmesi gerekir. Eğer bir hata yapılırsa da, yanlış parçaya kayıt atılması ya da yanlış parçadan kayıt çekilmesi engellenemeyen bir durum olacaktır.

Geleneksel veritabanlarının ilk problemi, kendi doğasında dağıtık yapıya sahip olmamasıdır. Bu yüzden veritabanı parçalama işlemleri gibi işlemlerde geleneksel veritabanları yardımcı olamamaktadır. Veritabanını işletmenin ve de uygulama kodunu değiştirmenin karmaşası uygulama geliştiricilerinin sorumluluğundadır.

Ayrıca sistem insan hataları için tasarlanmamıştır. Sistem daha da kompleks bir hale geldiğinde, hata yapma olasılığı artacaktır.

1.2. Büyük Veri Sistemlerinde İstenilen Özellikler

Veri sistemleri, geçmişten bugüne kadar kazanılan bilgiler üzerinden sorulara cevap veren sistemlerdir. Veri sistemleri bilgileri yalnızca kaydeden ve kaydettiği veriler üzerinden sonuç döndüren sistemler değildirler. Sonuçları üretmek için bilgileri farklı bilgilerle birleştirirler. Örnek vermek gerekirse, banka hesap bakiyesi, o hesaptaki tüm işlemleri birleştirerek hesaplanmaktadır.

Veri sistemlerinde tüm bilgiler eşit değildir. Bazı bilgiler diğer bilgilerden türetilmektedir. Sosyal medyada arkadaş sayısı bilgisinin, arkadaş listesi bilgisinden üretilmesi bu duruma örnektir. Bilginin en ham haline "veri" denir. Veri kelimesi genelde bilgi kelimesi ile aynı anlamda gibi kullanılmaktadır. Veri, tüm bilgilerin üretildiği özel bilgidir. [1]

Veri sistemleri, daha önce kaydedilmiş veriler üzerinden sorgu sonuçlarını döndürdüğünden, bu sistemlerin genel amacı, tüm veri setine bakarak sorulara cevap vermektir. Bu yüzden veri sistemleri;

Sorgu = Fonksiyon (Tüm Veri)

şeklindeki, bir tanımlama ile ifade edilebilir. Verilerle yapılabilecek her türlü işlem topluluğuna, sahip olunan verileri girdi olarak alan bir fonksiyon olarak ifade edilebilir.

1.2.1. Sağlamlık ve hata toleransı

Dağıtık sistemlerin zorlukları karşısında doğru çalışan sistemler oluşturmak kolay değildir. Sistemlerin; rastgele makineler çöktüğünde doğru davranması, verilerin mükerrer olmasını engellemesi, verilerin aynı anda erişimine izin vermesi gibi birçok zorunluğu mevcuttur. Güçlü bir büyük veri sistemi oluşturmanın bir parçası bu karmaşalardan kurtulmaktır [1,3].

Sistemlerin doğru çalışabilmesi için insan hatalarına toleransı yüksek olmalıdır. Çalışan bir sistemde insan hataları kaçınılmazdır. Uygulama kodunun yanlış bir zamanda yayınlanması ya da veritabanındaki değerlerin değiştirilmesi, verilerin bozulmasına neden olabilir. Eğer büyük veri sistemlerinin temeline değişmezlik yerleştirilirse; sistemler, kolay ve açık bir kurtarma mekanizması ile insan hatalarına kalıtsal olarak dayanıklı olmaktadır [1].

1.2.2. Güncelleme ve okumalarda düşük gecikme

Uygulamaların çok büyük çoğunluğunun, okuma işlemlerini birkaç milisaniye gibi çok düşük bir gecikme ile gerçekleştirmesi gerekmektedir. Diğer bir yandan, güncellemelerin ne kadar bir gecikme ile gerçekleşeceğinin gerekliliği, uygulamalar arasında değişiklik göstermektedir. Bazı uygulamalarda güncelleme işlemlerinin hemen gerçekleşmesi gerekmekte iken, bazı uygulamalarda güncelleme işlemlerinin birkaç saat içinde gerçekleştirilmesi yeterlidir. Gerçek zamanlı veri sistemlerinde, güncellenen verinin anlık olarak servis edilmesi gerektiğinden, okuma ve güncelleme işlemlerinin düşük bir gecikme ile gerçekleşmesi gerekmektedir. Daha da önemlisi, bunu sistemin sağlamlığını riske atmadan yapmak gerekmektedir.

1.2.3. Ölçeklenebilirlik

Ölçeklendirme; veri artışı ya da veri yükünün artması gibi durumlarda, uygulamanın performansının düşmesine izin vermememe yeteneğidir. İlişkisel veri tabanları ölçeklendirme için tasarlanmamışlardır. İlişkisel veritabanları sıralı, düzenli ve küçük boyuttaki verileri saklamak için tek bir sunucu üzerinde çalışmak için tasarlanmışlardır. Bu yüzden büyük miktarlardaki verilerin tek bir ilişkisel veritabanı üzerinde saklanması ve buradan servis edilmesi çok zordur [1,4].

1.2.4. Genellik

Genel bir sistem, çok çeşitli uygulamalara çözüm sağlayabilir niteliktedir. Uygulama hangi amaç için yazılmış olursa olsun ya da hangi veri tiplerine sahip olursa olsun, hazırlanacak veri sistemi her tür uygulamaya uygulanabilmelidir.

1.2.5. Genişletilebilirlik

Genişletilebilirlik, sistemlerin işleyişinde bir değişiklik yapıldığında ya da sistemlere yeni bir özellik eklendiğinde, sistemlerin tekrar dizayn edilmesine gerek kalmamasıdır. Genişletilebilir sistemler, fonksiyonalitelerin küçük bir geliştirme maliyeti ile eklenmesine izin vermektedirler.

Çoğu zaman yeni bir özelliğin eklenmesi ya da var olan bir özellik üzerinde değişiklikler yapılması, eski formatta olan verilerin yeni formata taşınmasını

gerektirmektedir. Genişletilebilir sistemler oluşturmanın bir kısmı, büyük ölçekli taşınmaların da kolaylıkla yapılabilmesidir [1].

1.2.6. Anlık sorgular

Veriler üzerinde anlık sorgulama yapabilmek gerçek zamanlı sistemler için oldukça önemli bir işlevdir. Sistem tarafından yapılan sorgulara hızlı ve doğru cevap dönülmesi, sistemin tutarlılığı bakımından oldukça önemlidir [1].

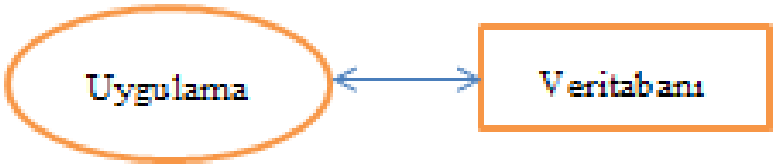
1.2.7. Minimal bakım

Bakım; bir sistemin düzgün bir şekilde çalışmasını ve devamlılığını sağlamak için gerekli olan iştir. Uygulamanın bakımı, geliştiricileri için bir yüktür. Bakım, üretimde olup yanlış giden herhangi bir hatayı ayıklamak, işlemlerin ayakta ve çalışmasını sağlamak, ölçeklendirmek için yeni makinelerin ekleneceği senaryoları geliştirmek gibi birçok işi kapsamaktadır.

Bakımı en aza indirgemenin önemli bir parçası, mümkün olduğunca karmaşası en az olan bileşenleri seçmektir. Genelde, dağıtık sistemler anlaşılması güç yapılara sahip olmaya meyillidirler. Sistem ne kadar kompleks olursa, sistemde hata olma olasılığı artmaktadır [1].

1.3. Artımlı Mimarilerde Yaşanan Problemler

Artımlı mimariler; yazma ve okuma işlemlerini kullanan ve sisteme yeni veri gelir gelmez veritabanına yazma, veritabanından okuma ve veritabanındaki durumun sağlanması gibi işlemlerin artış göstererek devam etmediği sistemlerdir. Artımlı mimariler uygulama doğrudan bağılıdırlar. Bu durum Şekil 1.2'de gösterilmektedir.



Şekil 1.2 Artımlı mimariler

Artımlı mimariler çok yaygın olarak kullanıldığı için, birçok uygulama geliştiricisi problemlerini başka bir mimari çözümü ile çözebileceğini fark edememektedir. Bu mimarilerin kompleksitesi köklenmiş olduğundan, geliştiriciler bu sorunlardan kurtulabilmek için bir yol aramamaktadırlar [1].

1.3.1. İşlevsel karmaşa

Veritabanlarına yazma/ veritabanlarından okuma işlemlerinde, bir disk indeksi eklendiğinde veya değiştirildiğinde, indeksinin bazı kısımları kullanılmaz olmaktadır. Oluşan boşluklar, diskin dolmasını engellemek için yeniden kullanılabilir hale getirilmelidirler. Bu işleme sıkıştırma denmektedir. Kullanılmayan kısımların, oluşur oluşmaz yeniden kullanılabilir yapılması oldukça masraflı bir işlemdir. Bu yüzden sıkıştırma işlemi belirli periyotlarla yapılmaktadır.

Sıkıştırma oldukça yoğun bir işlemdir. Sıkıştırma sırasında sunucunun CPU ve disk gereksinimi oldukça fazla olduğundan, sıkıştırma sırasında sunucunun performansında bir düşüş yaşanacaktır. Tüm sunucuların yoğun çalıştığı bir sistemde performans düşüşü, sunucuların kademeli olarak çökmesine neden olabilmektedir. Eğer aynı anda çok fazla makine sıkıştırılırsa; onların üzerindeki yük, bulutta bulunan diğer makinelere dağıtılmalıdır. Bu durum bulutun geri kalanına aşırı yükün yüklenmesine, sistemin tamamının çalışamaz hale gelmesine neden olabilmektedir.

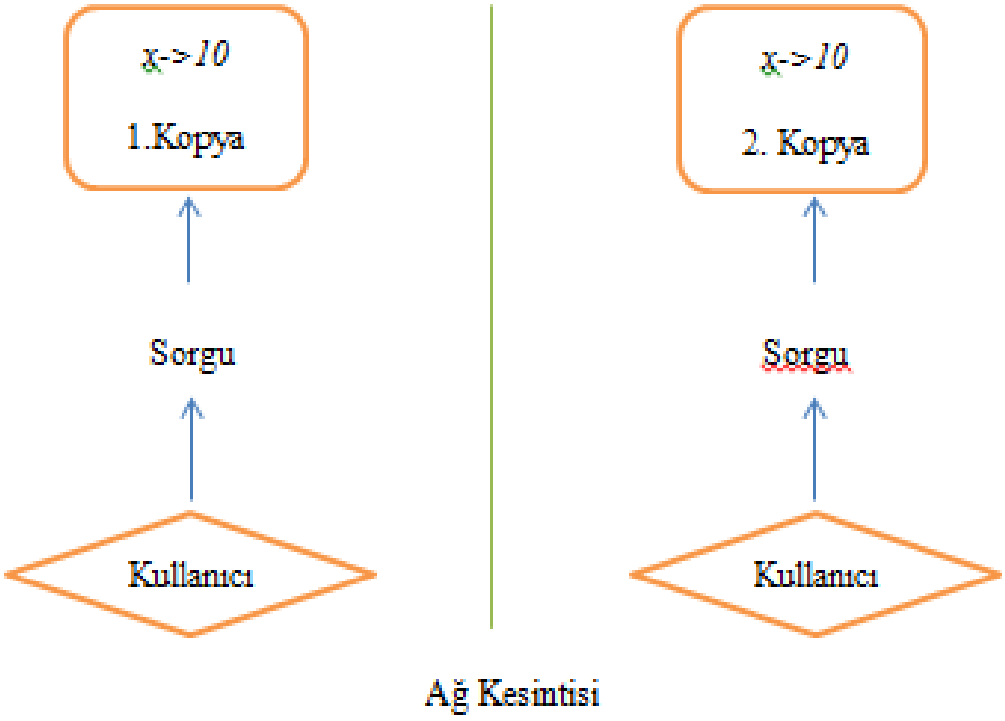
Sıkıştırma işleminin doğru olarak yapılabilmesi için sıkıştırmalar tek seferde çok fazla düğümün etkilenmemesini sağlayacak şekilde programlanmalıdır. Devam eden sıkıştırma sırasında başka düğümlerin eklenmesini engellemek için, bir sıkıştırmanın ne kadar sürdüğüne dikkat edilmelidir. Ayrıca sıkıştırma yapılan makine kullanılmadığında bulutun kapasitesinin yükü kaldıracağından emin olunmalıdır. Bütün bu işlemler geliştiriciler tarafından yapılabilmektedir ama bir karmaşadan kurtulmak daha iyi bir çözümdür [1].

1.3.2. Sistem tutarlılığının sağlanması

Artımlı mimarilerin diğer bir zorluğu ise sistemin yüksek erişilebilirlikte olması durumudur. Yüksek erişimli sistemler; ağın bir kısmında ya da herhangi bir makinede hata olduğunda dahi sorgu ve güncelleme işlemlerine izin veren sistemlerdir.

Erişilebilirlik, sistemler için önemli olan diğer bir özellik tutarlılık ile mukayese edilmektedir. Tutarlı sistemler; önceki tüm bilgileri göz önüne alarak sonuç döndürmektedirler. CAP teoremi, ağ bölünmelerinin olduğu bir sistemde erişilebilirlik ve tutarlılık kavramlarının bir arada bulanamayacağını göstermektedir. Bu yüzden erişilebilirliği yüksek olan sistemler ağ bölümü sırasında eski sonuçlar döndürebilmektedirler [1,5].

Dağıtık veritabanları, yüksek erişilebilirliği sağlamak için tüm verinin birçok kopyasını saklamaktadırlar. Aynı verinin birden fazla kopyasının olması, makinelerden biri çöktüğünde ya da ağ kesintiye uğradığında veriye erişimi sağlamaktadır. Şekil 1.3'te bir örnekleme görülmektedir. Bir ağ kesintisi sırasında, yüksek erişilebilir olan sistemlerde veri güncelleme talebinde bulunan kullanıcıları olabilir. Bu durum kopyaların farklılaşmasına ve farklı güncelleme seti almasına neden olabilir. Yalnızca ağ kesintisi bittiğinde kopyalar ortak bir değerde birleştirilebilir.



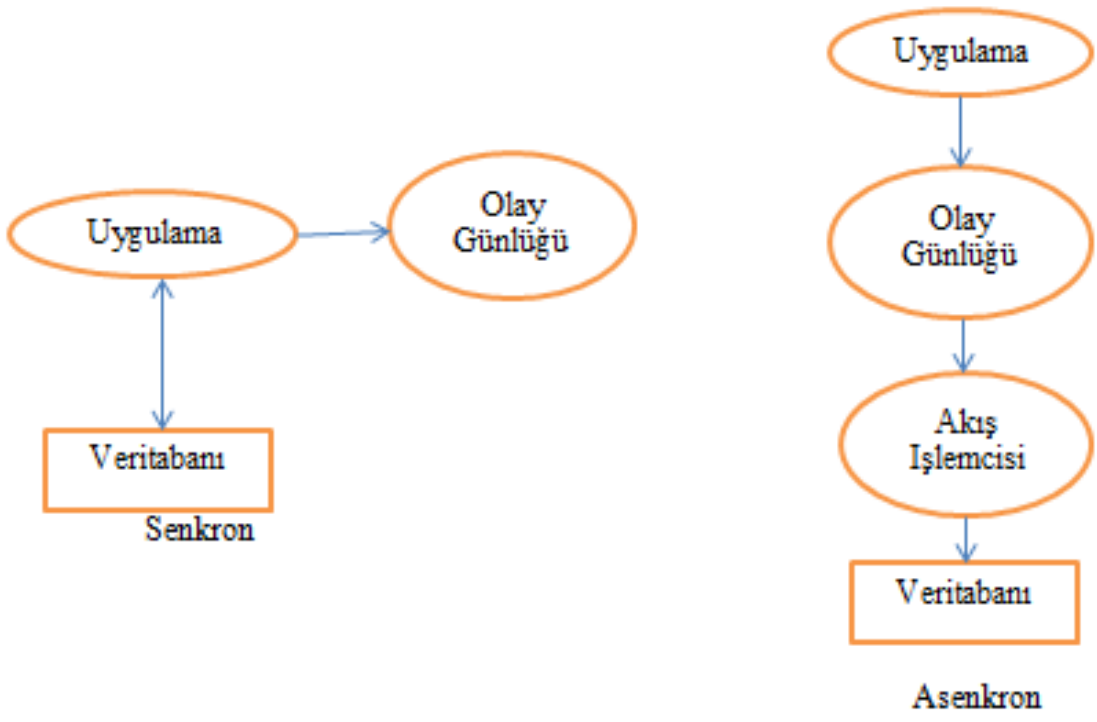
Şekil 1.3. Erişilebilirliği artırmak için kopya kullanımı

İki ayrı kopyada, veritabanında bir talep olduğunda değeri artış gösteren x değeri saklanmaktadır (Bkz. Şekil 1.3). Ağ kesintisi olduğunda; kullanıcılar tarafından sunuculara farklı sayıda talep olduğunda x değeri her iki kopyada farklı olacaktır. Ağ kesintisi bittiğinde verinin tutarlılığı için bu değerler birleştirilmelidir. Verinin doğru olması için bu birleştirme işleminin doğru yapılması gerekmektedir. Bu yüzden yalnızca x değerini tutmak yeterli olmayacaktır. Ağ kesintisi bittiğinde değerleri düzeltecek kod yazılımına ve veri değiştiğinde ona cevap verebilecek veri yapısına ihtiyaç vardır.

Genel olarak, artımlı ve yüksek erişilebilirliği olan sistemlerde tutarlılığı sağlamak kolay değildir ve bu karmaşa bu sistemlerin tabiatında vardır.

1.3.3. İnsan hata toleransı

Artımlı mimarilerdeki diğer bir sorun da, insan hatasına olan tolerans düşüklüğüdür. Artımlı sistemlerde veritabanındaki veriler değiştirilerek saklanmaktadır. Bu yüzden bir hata olduğunda veritabanındaki gerçek veri bozulabilmektedir. Hatalar kaçınılmaz olduğu için verinin bozulma riski çok yüksektir. Ama bu sorun, tüm mimariyi tekrar gözden geçirmeden çözülebilecek bir sorundur.



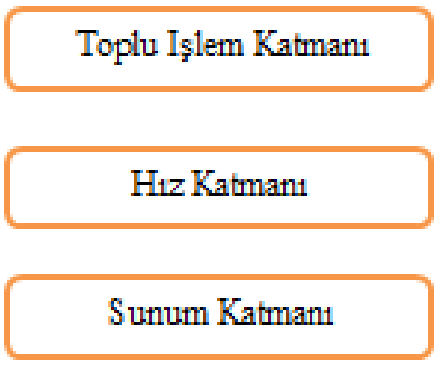
Şekil 1.4. Artımlı mimarilerde günlük ekleme

Şekil 1.4'te iki farklı mimari görülmektedir. Senkron mimarilerde uygulama, veritabanı güncellemelerini direk olarak yapmaktadır. Asenkron mimarilerde ise işlemler veritabanına gitmeden önce arkada bir kuyrukta bekletilmektedir. Her iki mimaride de her işlemin kaydı kalıcı olarak olay günlüğünde tutulmaktadır. Herhangi bir insan hatası olup veri bozulduğunda, olay günlüğünde kayıtlı olan işlemlere bakılarak, bozulan veriler tekrar doğru hale getirilmektedir. Olay günlüğü artımlı mimarilerde insan hatalarına çözüm sağlasa da, artımlı mimarilerin diğer sorunları hala önemliliğini korumaktadırlar [1].

2. LAMBDA MİMARİSİ

Sıralı olmayan bir veri seti üzerinde gerçek zamanlı rastgele işlemler yapmak oldukça zor bir problemdir. Bütün bir çözümü sağlayan tek bir araç bulunmamaktadır. Bu yüzden bir büyük veri sistemi tasarlarken çeşitli araçları kullanmak gerekmektedir [1].

Lambda mimarisi; büyük veri sistemlerini Şekil 2.1'de de görülen üç katmana ayırarak kurgulamaktadır. Her bir katman kendi işlevselliğini gerçekleştirmektir.



Şekil 2.1 Lambda mimarisindeki katmanlar

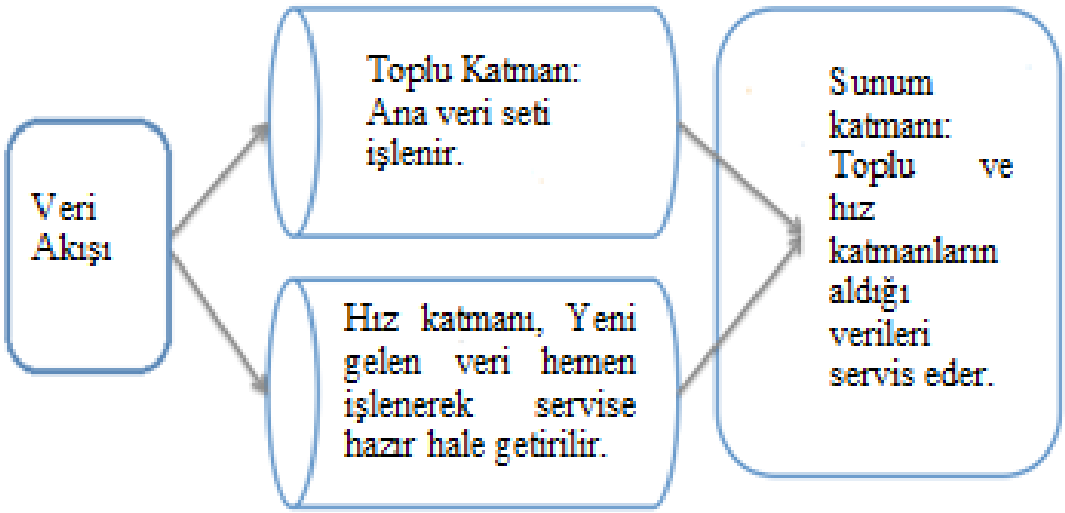
Lambda mimarisi, herhangi bir veri seti üzerinde çalışan, sorulara düşük bir gecikme oranı ile cevap döndüren bir fonksiyon tasarlamaktadır. Veritabanı tasarlayıcısının istekleri doğrultusunda spesifik teknolojiler kullanılabilir. Ama Lambda Mimarisi, bu teknolojileri seçecek, talepleri karşılamak için onları bir arada kullanacak bir yaklaşım sunmaktadır [1].

Ölçeklendirme, mimaride kullanılacak dağıtık yapıdaki araçlar ile kolaylıkla sağlanabilmektedir. Mimari, sistemler için genel bir çözüm sunmaktadır. Finans yönetim uygulamaları, sosyal medya analiz sistemleri, coğrafi bilgi sistemleri gibi tüm uygulama türlerini genelleştirerek çözüm sunmaktadır.

Lambda mimarisi, sistemin karmaşasını temel bileşenlerden uzak tutup, birkaç saat sonra çıktılarının öneminin kalmadığı parçalara dağıtmaktadır. Okuma/ yazma işlemlerinin yapıldığı dağıtık veritabanları gibi karmaşık bileşenler, sonuçları kısa bir süreliğine önemli olan katmanda bulunmaktadır. Artımlı mimarilerde anlık sıkıştırma işlemleri bir zorluk iken, Lambda mimarisinde ana veri tabanları herhangi bir anlık sıkıştırma istememektedir.

2.1. Büyük Veri İçin Veri Modeli

Lambda mimarisinin merkezinde veri doğruluğunun kaynağı olan asıl veri seti yer almaktadır. Sunum ya da hız katmanındaki herhangi bir veri kaybedilse bile asıl veri setinden tekrar üretilebilir. Şekil 2.2'de Lambda mimarisinde verinin işleyişi görülmektedir.



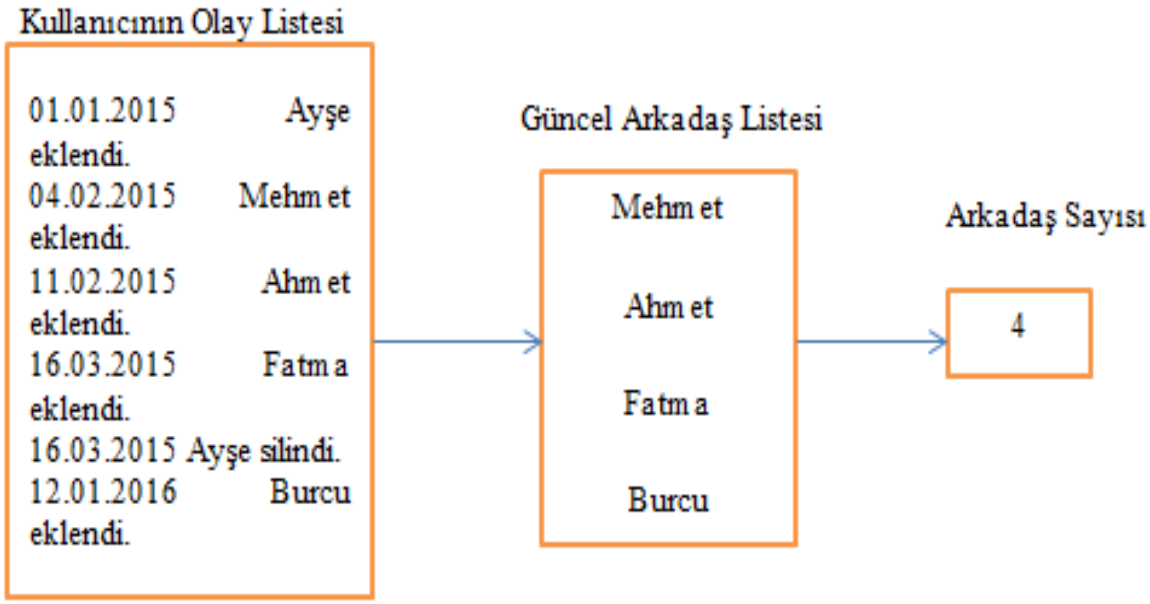
Şekil 2.2 Lambda Mimarisinde verilerin işleyişi.

Ana veri seti, Lambda mimarisinin bozulmadan korunması gereken tek kısımdır. Makinelerin aşırı yüklenmesi, disklerin çökmesi gibi risklere karşı sistem dikkatlice tasarlanmalıdır. Asıl veri setinin iki bileşeni vardır. Bunlar: Kullanılan veri modeli ve asıl veri setinin saklanılmasıdır.

2.1.1. Verinin özellikleri

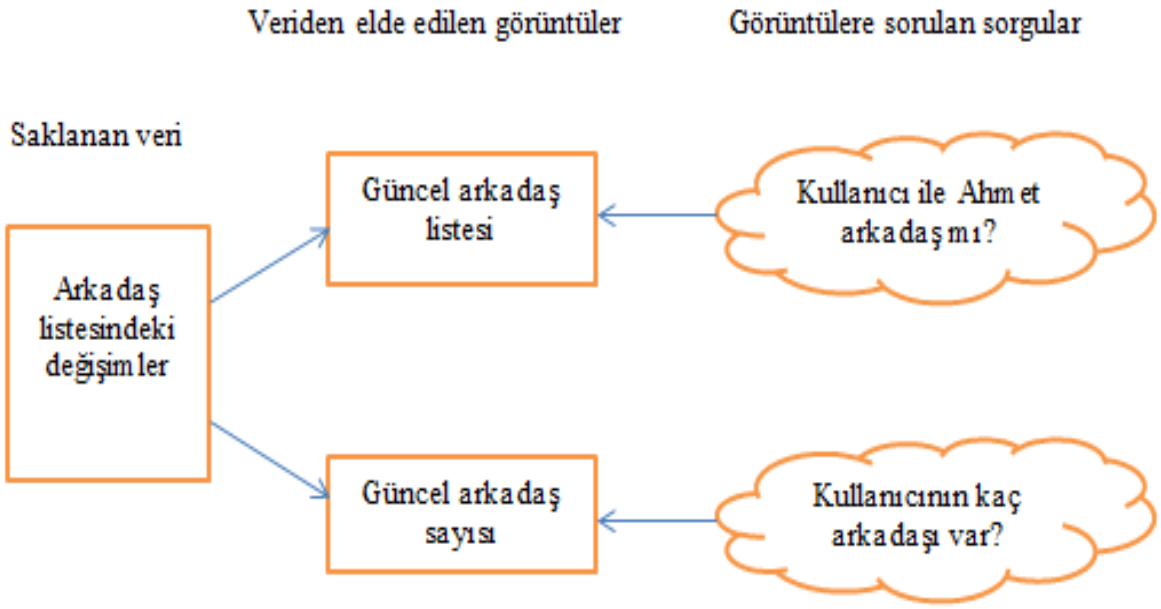
Lambda mimarisi için tasarlanan veri modelini daha iyi anlamak için Şekil 2.3'te gösterildiği gibi bir örnekleme yapılabilir: Facebook benzeri bir sosyal medya uygulaması yazılmaktadır. Yeni bir kullanıcı siteye katıldığında, kullanıcı arkadaşlarını ve ailesini davet etmeye başlamaktadır. Kullanıcının bağlantılarının nasıl saklanacağına karar vermek gerekmektedir. Kullanıcının arkadaşlarıyla ve tanımadığı kişilerle olan olaylar, kullanıcının güncel arkadaş listesi ya da kullanıcının güncel arkadaş sayısı veritabanında veri olarak tutulacak bilgiler olabilir. Yalnızca güncel arkadaş listesinin sayısı tutulduğunda, kullanıcının arkadaşları hakkında detaylı bilgi alınamaz. Güncel arkadaş listesi tutulduğunda ise, kişinin geçmiş arkadaşları hakkında bilgi alınamaz. Bu yüzden

kullanıcının olaylar listesinin tutulması, kullanıcı hakkında en geniş bilginin elde edilmesini sağlamaktadır.



Şekil 2.3 Sosyal Medya Uygulama örneğinde bilgilerin oluşturulması.

Bu yapıda; kullanıcının arkadaşları ile ilgili bilgiler sol tarafındaki bilgiden üretilmektedir. Bu işlem tek yönlü olarak yapılmaktadır.



Şekil 2.4 Veri, görüntü ve sorgular arasındaki ilişkiler

Şekil 2.4'te veri, görüntü ve sorgular arasındaki ilişki görülmektedir. Görüntüler saklanan işlenmemiş veriden elde edilmektedir. Uygulamaya sorulan sorgular bu görüntüler üzerinden cevaplanmaktadır.

Lambda mimarisinde kullanılan veri terimleri aşağıdaki gibidir:

- Bilgi: Büyük veri sistemiyle alakalı bilinen her şeydir.
- Veri: Her hangi bir şeyden üretilemeyen bilgi olarak adlandırılmaktadır.
- Sorgu: Veriye sorulan sorulardır.
- Görüntü: Asıl veriden üretilen bilgilerdir. Özel tipteki sorulara cevap vermek için oluşturulmaktadır.

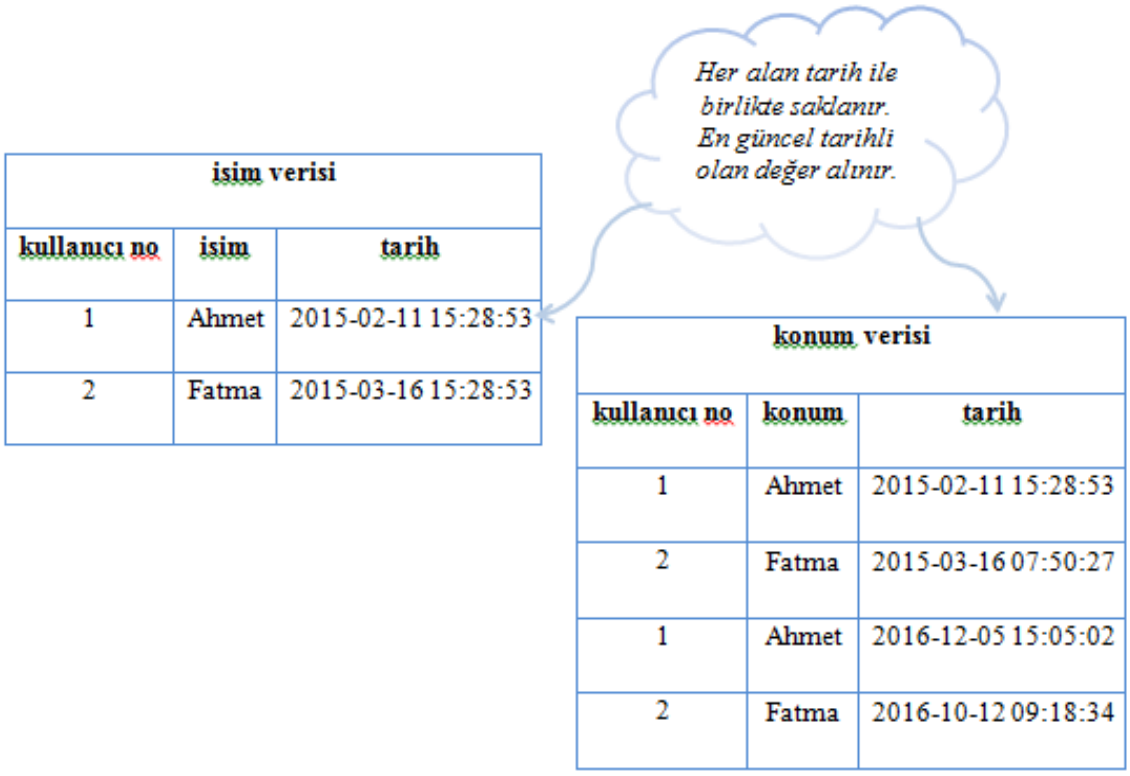
Aşağıda Lambda mimarisi için tasarlanan veri modelinin özellikleri sıralanmaktadır:

Verinin İşlenmemişliği: Veri sistemleri; geçmişte kaydedilen veriler hakkında sorulan sorulara cevap veren sistemlerdir. Sistemin cevap vereceği sorular önceden belli olmadığı için, saklanan verilerin işlenmemişliği oldukça değerlidir. Bu yüzden verinin ne kadar işlenmemiş hali saklanırsa, tasarlanan sistem de o kadar fazla soruya cevap verebilmektedir. Verinin özetlenerek, üzerinde yazılarak ya da silinerek saklanması, verilerden çıkarılacak soruları azaltmaktadır.

Verinin Değişmezliği: Değişmez veri kavramı, ilişkisel veritabanları için oldukça uzak bir konsepttir. Birçok veritabanında, güncelleme veri tabanının gerçekleştirdiği esas işlemlerden biridir. Oysa veri güncellenmez ya da silinmez; daha fazla veri eklenir. Büyük veri sistemleri için değişmez veri modeli kullanmak iki önemli yarar getirmektedir. Bunlardan ilki insan hata toleransıdır. Değişmez veri modelinin en önemli avantajı insan hata payını ortadan kaldırmasıdır. Diğer veri modellerinde kullanıcı bir hata yaptığında veriyi kurtarmak için farklı mekanizmalara sahip olmak gerekmektedir. Ama bu veri modelinde veri kaybı olmadığı için, doğru veriye tekrar ulaşmak için hatalı verileri silmek, ana veri setinden görüntüleri tekrar oluşturmak yeterlidir. Diğer yarar ise basitliktir. Değişebilir veri modellerinde bir nesneyi güncelleyebilmek ya da o nesneye erişebilmek

için verinin indekslenmesi gerekmektedir. Bu veri modelinde ise sistemin tek yapması gereken yeni veri ekleyebilme yeteneğinin olmasıdır [1].

Verinin değişmezliğini anlayabilmek için sosyal medya uygulaması örneği düşünülebilir (Bkz. Şekil 2.4). Kullanıcı yaşadığı konumun bilgisini de profilinde saklamaktadır. Değişmez veri modelinde her alan ayrı olarak tarih bilgisi ile birlikte saklanmaktadır. Eğer birbiri ile ilişkili alanlar var ise bu alanlar birlikte saklanmalıdır. Şekil 2.5'te isim ve konum verisinin saklanmasına örnek verilmiştir. 1 ve 2 numaralı kullanıcılar konumlarını değiştirdiklerinde, eski konum verileri silinmeden güncel konum bilgileri tarih ile eklenmektedir.



Şekil 2.5 Değişmez veri modelinde alanların saklanması

Verinin her zaman doğru olması: Değişmez veri modelinin en önemli sonucu verinin ömür boyu doğru olmasıdır. Bir kez doğru olan veri her zaman doğru olmalıdır. Genelde ana veri seti, yeni değişmez veriler eklenerek büyümeye devam etmektedir. Veri saklama stratejilerinde verinin ömür boyu doğruluğu ile çelişkili olan bazı özel durumlar olabilmektedir. Çöp toplama, ana veri setinin büyümesini kontrol altına almak için kullanılabilmektedir. Örnek vermek gerekirse; sosyal medya örneğindeki kullanıcı konum

bilgisi, her yıl yalnızca bir konum değişikliği saklanacak şekilde saklanabilir (Bkz. Şekil 2.3).

Değişmez veriyi silmek bu model için bir tezatlık gibi görünse de çok nadir durumlarda silme işlemi yapmak gerekebilir. Normal durumlarda veri değişmezdir. Değişmez veriyi silerken en üst seviyede bir dikkat ile doğru verinin filtrelendiğinden emin olduktan sonra silme işlemi gerçekleştirilmeli; veri eski versiyonuyla değiştirilmelidir [1].

2.2. Gerçek-tabanlı Veri Modeli

Veri, herhangi bir şeyden üretilemeyen bilgi setidir. Veri, ilişkisel veri tabanları tabloları dışında, XML ya da JSON olarak saklanabilmektedir. Gerçek tabanlı veri modeli bu yapılardan daha farklı olarak, veriyi gerçek diye adlandırılan birimlere ayırarak, bilgiyi tarih ile birlikte tutan veri modelidir.

Gerçek-tabanlı veri modelinde, her bir gerçek kendinden daha küçük anlamlı birimlere ayrılamaz. Gerçekleri birbirinden ayırabilmek için her bir gerçeğe birbirinden ayırabilecek ayırt edici anahtarlar eklenebilmektedir.

Kısaca gerçek-tabanlı veri modeli aşağıdaki üç madde ile tanımlanabilmektedir:

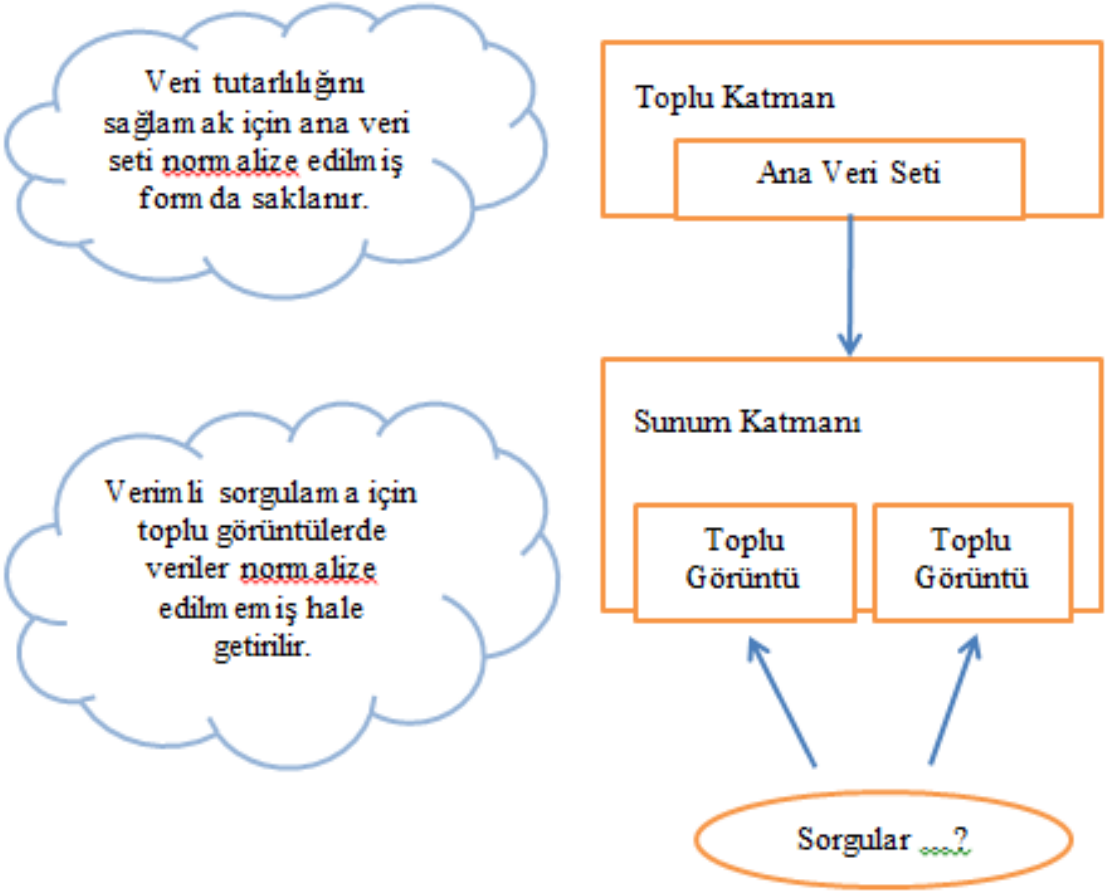
- İşlenmemiş veri gerçekler mantığıyla saklanmaktadır.
- Tarih kullanılarak gerçeklerin değişmezliği ve ebediyen doğruluğu korunmaktadır.
- Gerçekler bir ayırt edici anahtarlar ile özelleştirilerek, sorguların mükerrer verileri ayırt edebilmesi sağlanmıştır.

Gerçek-tabanlı veri modeli ile veriler tarihsel olarak sorgulanabilmektedir. Veri değişmez olduğu için, en güncel bilgiyi tutmak yerine bilginin tarihsel süreci saklanmaktadır.

Gerçek tabanlı veri modelinde, insan hataları tolere edilebilmektedir. Hatalı gerçekler silinerek verinin doğruluğu sağlanabilmektedir. Veriler, veritabanında parçalı bir şekilde tutulduğu için NULL değerleri kullanmadan veri seti oluşturulabilmektedir. Sütun mantığıyla saklanan verilerde; herhangi sütun değeri boş olduğunda, veri NULL değeri ile

birlikte saklanmaktadır. Bu modelde ise, her bilgi en küçük anlamlı bilgi mantığıyla tutulduğundan, NULL değerine ihtiyaç yoktur.

Veri modelini ilgilendiren diğer bir husus verilerin normalize edilmiş formda saklanıp saklanmayacağıdır. Normalizasyon verilerin tutarlılığını sağlamak ve veri fazlalığını minimize etmek için, verilerin yapısal bir şekilde saklanmasıdır. Normalize edilmiş veriler, ilişkisel tablolarda saklanmaktadır. İlişkisel veritabanları, sorgulara birbiri ile ilişkilendirilen tabloları birleştirerek cevap vermektedirler. Bu da sorgu sonuçlarının daha geç dönmesine neden olmaktadır. Veriler, normalize edilmeden saklandığında sorgu sonuçları daha hızlı dönse de, verilerin tutarlılığını sağlamak zordur. Bu yüzden veritabanları tasarlanırken sorguların daha verimli çalışmasının mı yoksa veri tutarlılığının sağlanmasının mı daha önemli olduğu tartılıp, bu iki şema arasında seçim yapılmalıdır [1].

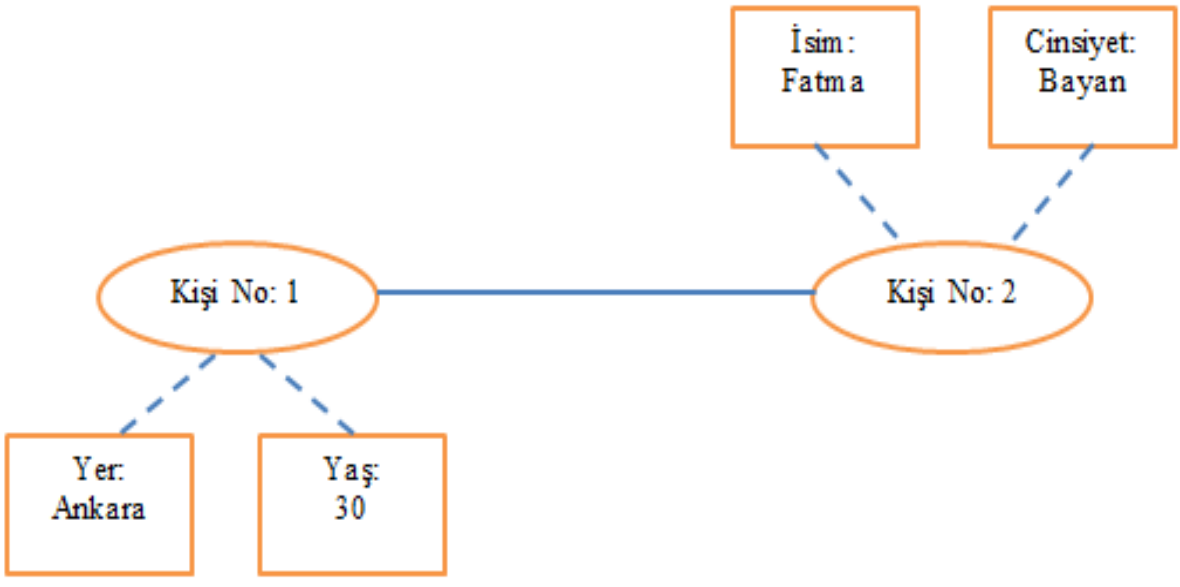


Şekil 2.6 Lambda Mimarisinde verileri saklama formları

Lambda mimarisi hem normalize edilmiş hem de edilmemiş verilerin sağlayacağı avantajları bir arada sağlamaktadır. Şekil 2.6'da da görüldüğü üzere, Lambda mimarisinde ana veri seti normalize edilmiş formda, ana veri setinden elde edilen toplu görüntüler ise normalize edilmemiş formlarda saklanmaktadır. Ana veri setinde olan herhangi bir veri parçası, birden fazla toplu görüntü içinde yer alabilmektedir. En önemli fark, toplu görüntülerin ana veri seti üzerindeki fonksiyonlar gibi tanımlanmasıdır. Bu yüzden toplu görüntülerdeki herhangi bir verinin güncellenmesine gerek yoktur. Bir toplu görüntü ana veri seti üzerinden tekrar oluşturulduğunda; güncellenmiş veri o toplu görüntüye yansıyacaktır. Bu yüzden Lambda mimarisinde toplu görüntüler ile ana veri seti uyumsuz olmayacaktır. Bu mimari, normalize edilmiş veriler üzerinde, indeksleme ile sorguların daha verimli çalışmasını sağlamaktadır.

2.2.1. Diyagram şemaları

İlişkisel veri tabanlarında, farklı tablolardaki verileri ilişkilendirmek için yabancı anahtar kavramı kullanılmaktadır. İlişkilendirilecek tablonun birincil anahtarı, ilişkilendirilen tablonun yabancı anahtarı ile bağlanmaktadır. İlişkilendirilen tablolar veritabanının genel mimarisini göstermektedir. Gerçek tabanlı mimaride veriler ilişkisel veri tabanlarında saklanmadığında, verileri ilişkilendirmek için diyagram şemaları kullanılır. Bir diyagram şemasının üç temel bileşeni bulunmaktadır. Bunlar; düğüm (node), kenar (edge) ve özellik (property) kavramlarıdır.



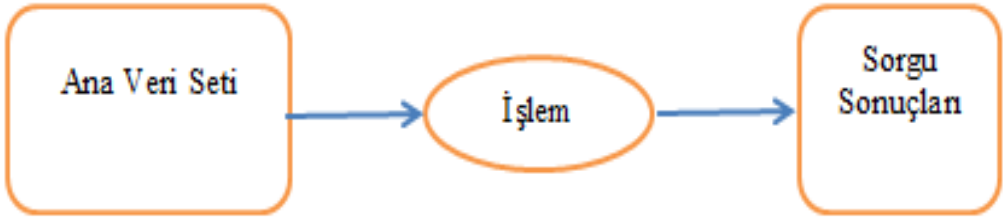
Şekil 2.7 Diyagram şemalarına örnekleme

Düğümler, sistemde tekil bir ayırt edici anahtar ile ifade edilen varlıklardır. Şekil 2.7'deki örneklemede görüldüğü üzere ayırt edici anahtar ile ifade edilen elipsler içindeki her bir kişi, sistemdeki bir düğümdür.

Kenarlar, Şekil 2.7'de düz çizgi ile gösterilen iki düğüm arasındaki ilişkilerdir. Birden fazla kenar türü oluşturularak, düğümler arasındaki ilişkiler çeşitlendirilebilmektedir. Özellikler, varlıklar hakkındaki bilgilerdir. Şekil 2.7'deki örnekte bulunan isim, yaş gibi kişisel bilgilerin hepsi özelliktir. Diyagram şemaları veri seti içindeki tüm verilerin bütünsel tanımını yapar [1].

2.3. Toplu Katman

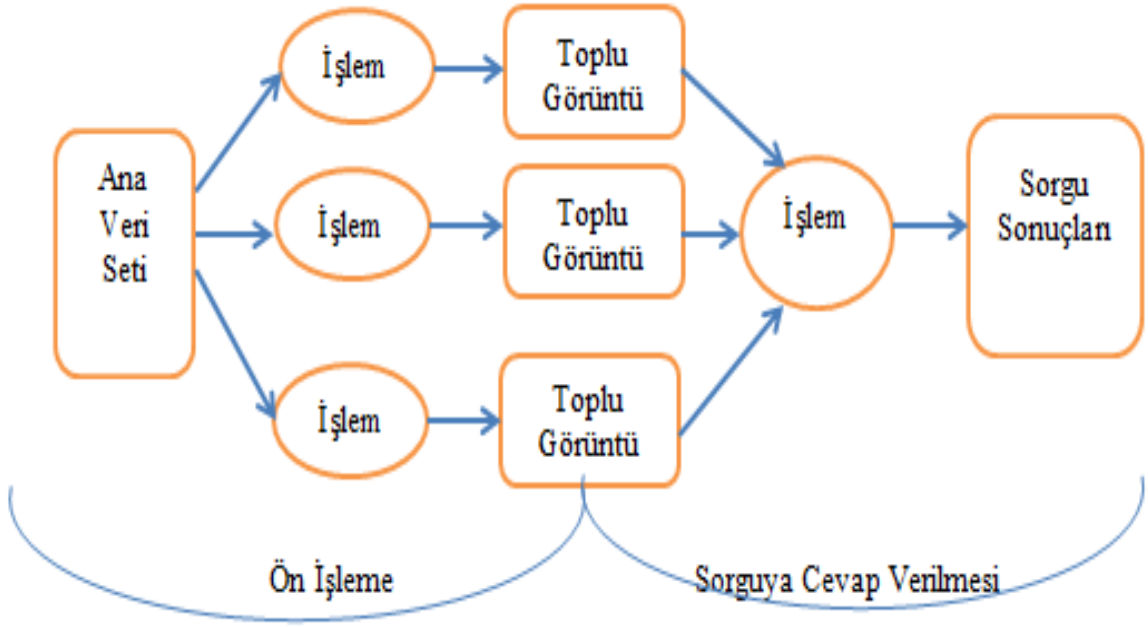
Toplu katman; toplu görüntüleri, ara verileri üretmek için ana veri seti üzerinde işlem yapan katmandır. Sunum katmanı, toplu görüntüleri indeksleyerek verilerin hızlı erişimine izin vermektedir.



Şekil 2.8 Ana veri setinin direk olarak işlenmesi

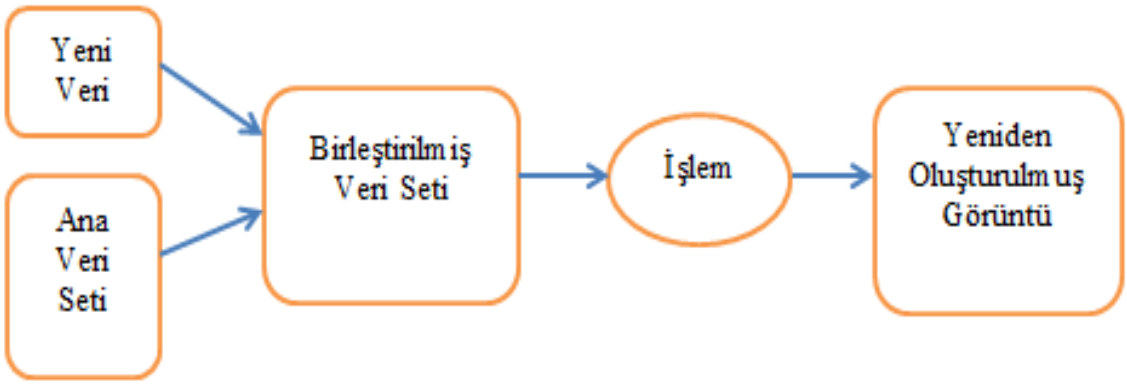
Toplu katmanda ana veri seti üzerinde yapılan işlemler zaman almaktadır. Bu durum olumsuzluk gibi görünse de, Lambda mimarisinde sorgulara hızlı cevap döndürülmesi, mimarideki diğer katmanlar tarafından sağlanmaktadır. Şekil 2.8'de toplu katmanda ana veri setinin işlenmesi için kullanılabilecek temel strateji görülmektedir. Bu stratejide, tüm olası sorgu sonuçları ön belleğe alınarak sunum katmanında aktarılmaktadır [1,6].

Olası her sorgunun cevabının toplu görüntülerde olması çok zordur. Bu yüzden Şekil 2.9'da görüldüğü gibi ana veri setinden ara sonuçlar oluşturup, bu sonuçları anlık sorgularda kullanarak sistemin hızlı cevap vermesi sağlanabilmektedir.



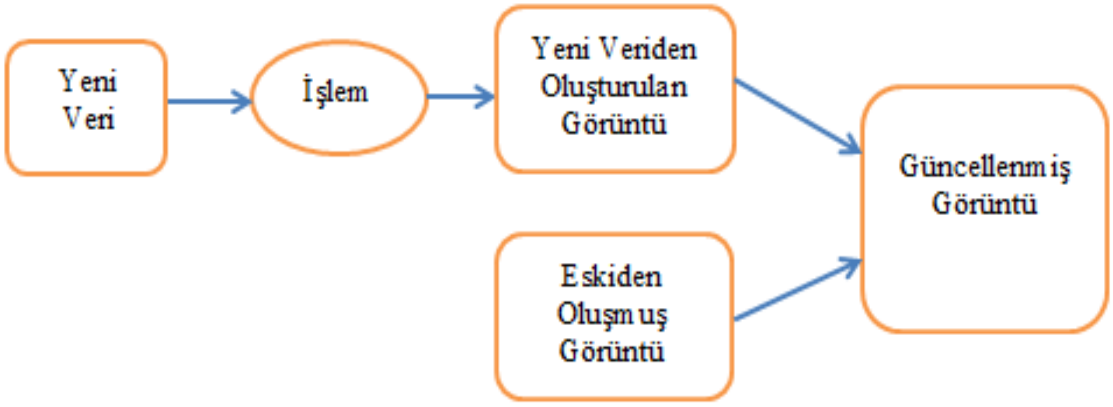
Şekil 2.9 Verilerin ön işlemden geçirilmesi ve sorgulara cevap verilmesi

Ana veri sürekli büyüdüğü için, yeni veri geldiği zaman toplu görüntülerin nasıl güncelleneceğine karar vermek gerekmektedir. Güncelleme işlemi için iki farklı algoritmadan birisi kullanılmaktadır. Bunlardan ilki, yeniden işleme algoritmasıdır. Yeni veri geldiği zaman, ana veri seti yeniden ön işlemden geçirilerek, tüm toplu görüntüler yeniden oluşturulmaktadır. Diğer seçenek ise artırımlı algoritmadır. Bu algoritmada ise yeni veri geldiği zaman toplu görüntüler direk olarak güncellenmektedir.



Şekil 2.10 Yeniden işleme algoritması.

Şekil 2.10'da yeniden işleme algoritması görülmektedir. Yeni gelen veri ana veri seti ile birleştirildikten sonra oluşan yeni veri seti işlemden geçerek yeni görüntüler oluşturulur.



Şekil 2.11 Artırımlı algoritma

Şekil 2.11'deki artırımlı algoritmada yeni gelen veri işlemden geçtikten sonra eski görüntüler güncellenmektedir.

Toplu katmanda kullanılacak olan algoritmaya karar verirken iki önemli unsur göz önünde bulundurulmalıdır. Bunlardan ilki, yeni gelen veri ile bir toplu görüntüyü güncellemek için ne kadar kaynağın gerektiğidir. Diğeri de, üretilen toplu görüntülerin büyüklüğüdür [1].

Artırımlı algoritma, güncelleme işlemi için yeni gelen veri ile ilgili toplu görüntülerin en son halini kullandığı için, yeniden işleme algoritmasına göre çok daha az kaynak kullanmaktadır. Yeniden işleme algoritması ana veri setinin tamamı üzerinden toplu görüntüleri oluşturacağından, artırımlı algoritmanın ihtiyaç duyacağı kaynaktan çok daha fazlasına ihtiyaç duymaktadır. Ama artırımlı algoritmada oluşturulacak toplu görüntülerin büyüklüğü yeniden işleme algoritmasında oluşturulacaklardan çok daha büyük olmalıdır. Çünkü artırımlı algoritmada, toplu görüntüleri artarak devam eden güncelleme işlemlerini yapabilecek boyutta olmalıdır [1].

Artırımlı algoritma, yeniden işleme algoritmasına göre hızlı çalışsa da, güncelleme işlemleri için yazılacak kodlar çok karışıktır. Bu da sistemin karmaşıklığını artırmaktadır.

Yeniden işleme algoritması işleyiş mantığı gereğince insan hatalarına toleranslıdır. Toplu görüntüler belirli aralıklarla ana veri setinden tekrar oluşturulduğundan; görüntüler üzerinde oluşacak hatalar görüntüler yeniden oluştuğunda ortadan kalkacaktır. Artırımlı algoritmalarda ise, insan hatalarını düzeltmek kolay değildir. Hataları düzeltmek için ayrıntılı olarak tutulan günlüklerden yararlanılabilmektedir ama günlüklerde bulunan bilgiler verileri düzeltmek için her zaman yeterli olmayabilmektedir. Çizelge 2.1'de artırımlı algoritma ve yeniden işleme algoritması karşılaştırılmıştır.

Çizelge 2.1. Artırımlı algoritma ve yeniden işleme algoritmasının karşılaştırılması [1]

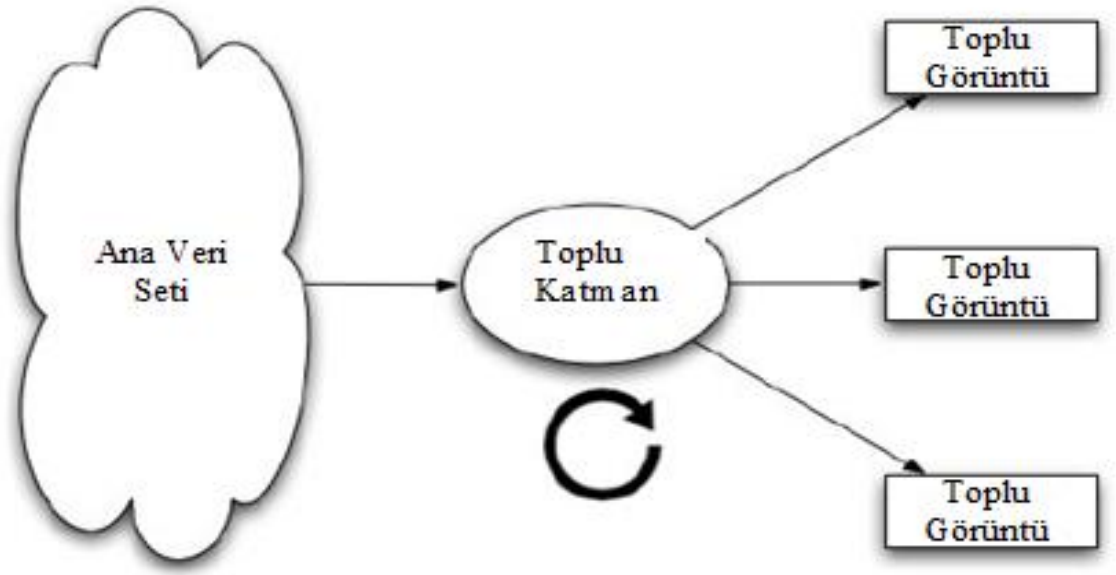
	Yeniden İşleme Algoritması	Artırımlı Algoritma
Performans	Tüm veri setini tekrar işlemek için işlem gücüne ihtiyaç vardır.	Daha az işlem gücüne ihtiyaç var ama oluşturulacak toplu görüntüler çok daha büyük olabilir.
İnsan Hata Toleransı	Toplu görüntüler yeniden oluşturduğu için insan hatalarına oldukça toleranslıdır.	İnsan hatalarını düzeltmek kolay değildir. Günlükler yetersiz kalabilir.
	Güçlü bir veri işleme sistemi tasarlamak için gereklidir.	Sistemin verimini artırabilir ama yeniden işleme algoritmalarına destek olarak kullanılmalıdır.

Ölçeklenebilirlik, daha fazla kaynak eklendiğinde sistemin doğru çalışmaya devam edebilme yeteneğidir. Büyük veri sistemlerinde veri yükü; uygulamaların cevap verdiği sorgu sayısı, her gün sisteme kayıt edilecek verinin miktarı ve sistemdeki toplam veri miktarının kombinasyonu ile ifade edilir. Veri sistemlerinin doğrusal olarak ölçeklenebilmesi gerekir. Doğrusal ölçeklenebilir sistemlerde veri miktarının artımı kadar kaynak artımı yapıldığında sistem aynı hızda toplu görüntüleri oluşturmalıdır. Örnek vermek gerekirse; ana veri seti iki katına çıkarıldığında, kullanılan sunucuların sayısının

iki katına çıkarılması, sistemin aynı hızda toplu görüntüleri oluşturması için yeterli olmalıdır.

2.3.1. Toplu katmanda veri saklama

Toplu katman, toplu görüntüleri oluşturmak için, ana ver setini işlemektedir. Bu yüzden toplu katmanda kullanılacak depolama sistemi tek seferde fazla miktarda datayı okuyabilecek kapasitede olmalıdır. Bu katmanda verilere rastgele erişimler yapılmasına gerek yoktur. Şekil 2.12'de toplu katmanın şeması görülmektedir.



Şekil 2.12 Toplu katman

Toplu katmanın verimli çalışabilmesi için seçilecek depolama sisteminin aşağıdaki özelliklere sahip olması gerekmektedir:

- Yeni gelen verilerin ana veri setine yazma işleminin kolay ve verimli bir şekilde yapılabilmesi gerekmektedir.
- Toplu katman ana veri setini sakladığı için, ana veri seti büyüdüğünde ölçeklendirmenin yapılabilmesi gerekmektedir.
- Toplu görüntüler ana verinin tamamı işlenerek üretilmektedir. Bu yüzden ana veri seti çok büyük ise; veriyi işleyebilmek için kullanılan sistem paralel işlemler yapabilmelidir.

- Verilerin değişmezliğini sağlayabilmek için, verileri değiştirebilecek işlemlere izin verilmemesi gerekmektedir. Toplu katmanında kullanılacak olan sistemde, var olan verinin ezilmesini önlemek için kontroller yapılmalıdır. [1].

2.3.2. Toplu katmanda kullanılacak depolama aracının seçimi

Aşağıda toplu katmanda kullanılabilecek veri depolama araçlarının tipleri açıklanmıştır:

- Anahtar / Değer (key/value) Depolama Sistemleri :Anahtar değer depolama sistemleri, birçok dilde bulunan hashmap mantığı ile çalışmaktadır. Veriler anahtar değerler ile birlikte saklanmaktadır. Eğer veri seti anahtar değer depolama sisteminde saklanacak ise ilk yapılması gereken anahtar değerinin ne olacağına karar vermektir. En uygulanabilir olanı anahtar değer olarak UUID (universally unique identifier, evrensel benzersiz tanımlayıcı) kullanmaktır. Anahtar-değer depolama sistemleri verileri yapılandırılmış formatta saklamaz. Hbase, Redis, Cassandra, Big Table gibi platformlar anahtar/değer depolama sistemleridir [7].

- Doküman (Document) Depolama Sistemleri :Doküman depolama sistemleri anahtar-değer depolama sistemlerine benzerdirler. Doküman depolama sistemleri XML/ JSON gibi veri tipleri kullanarak veriyi yapılandırılmış formatta saklamaktadırlar. Veriler veri tabanında dokümanları simgeleyen benzersiz anahtarlar ile tutulurlar. Dokümanlar yapılandırılmış bir format kullansalar da, çoğu zaman veriler için sabit şema tanımları kullanmazlar. Bu yüzden karmaşık dokümanlar ve çeşitli veri yapıları aynı veritabanında saklanabilir. Dokümanların özellikleri indeksleme ve sorgulama için, kullanıldığından anahtar- değer veri sistemlerine göre daha kompleks sorgulara izin vermektedirler. MongoDB, Couchbase, Amazon DynamoDB ve CouchDB doküman depolama sistemlerinin önemli örnekleridir [1,8].

- Dağıtık Dosya Sistemleri: Dosya sistemleri, bayt dizileri olan dosyaları sıralı bir şekilde disk üzerinde saklayan sistemlerdir. Dosyalar bazen bloklara ayrılabilir ama okuma ve yazma işlemi gerçekte sırala olarak yapılmaktadır. Sistem geliştiricisi, dosya sistemlerinde bir dosyadaki her bayt üzerinde tam kontrole sahiptirler. Sıradan bir dosya sisteminin sorunu, tek bir makine üzerinde çalışmasıdır. Bu yüzden bu tür dosya sistemlerinde yalnızca kullanılan tek makinenin kapasitesine göre ölçeklendirilme

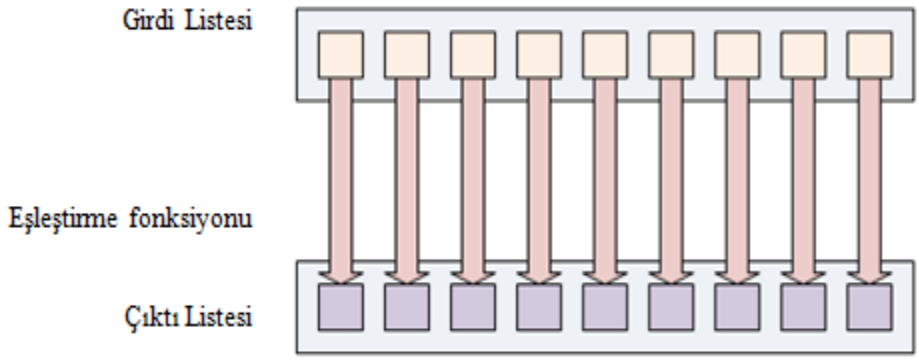
yapılabilmektedir. Dağıtık dosya sistemleri, üzerlerindeki verileri bilgisayarlar kümesine dağıtan dosya sistemleridir. Dağıtık dosya sistemleri, makinelerden birisi çöktüğü zaman hata toleransını sağlayabilmek ve verilere erişimi sağlamak üzere tasarlanmıştırlar [1].

- İlişkisel Veri Tabanları: İlişkisel veri tabanları; verileri sütun ve satırlardan oluşan tablolarda saklayan veri depolama sistemleridir. İlişkisel veri tabanlarında, tablolardaki verileri anahtarlar ile ilişkilendirilir. SQL Server, Oracle, PostgreSQL ve MySQL ilişkisel veritabanlarının önemli örneklerindendir.

Toplu katmanda kullanılacak depolama aracının sağlaması gereken asgari özellikler Bölüm 2.3.1'de belirtilmiştir. Birçok dağıtık veritabanı bu özellikleri sağlamaktadır. Dağıtık veri tabanı, fiziksel olarak farklı yerlerde olan, birbirlerine bir ağ ile bağlanmış veri tabanlar koleksiyonudur. Dağıtık veri tabanlarından birçoğu toplu katman için kullanılabilir ama eğer ana veri seti içerisinde bilgiler arasında çok fazla ilişki var ise, dağıtık veri tabanlarını kullanmak oldukça fazla iş yükü getirebilmektedir.

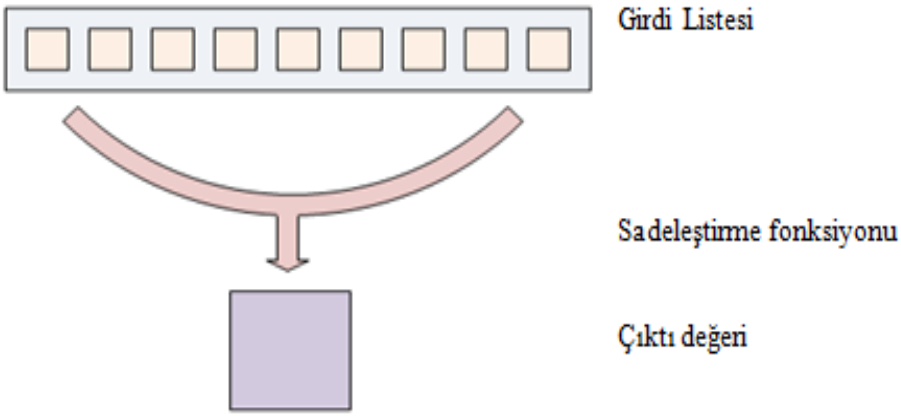
2.3.3. MapReduce: büyük verileri işlemek için paradigma

MapReduce, Google tarafından duyurulan, ölçeklenebilir ve hata toleransına sahip bir şekilde büyük miktardaki verileri parçalara ayırarak paralel olarak işleyebilmeye olanak sağlayan bir dağıtık işleme paradigmasıdır. Büyük miktardaki veriler "eşleştirme (map)" tarafından birbirinden bağımsız yığınlara ayrılır. Şekil 2.13'te eşleştirme işlemi görülmektedir.

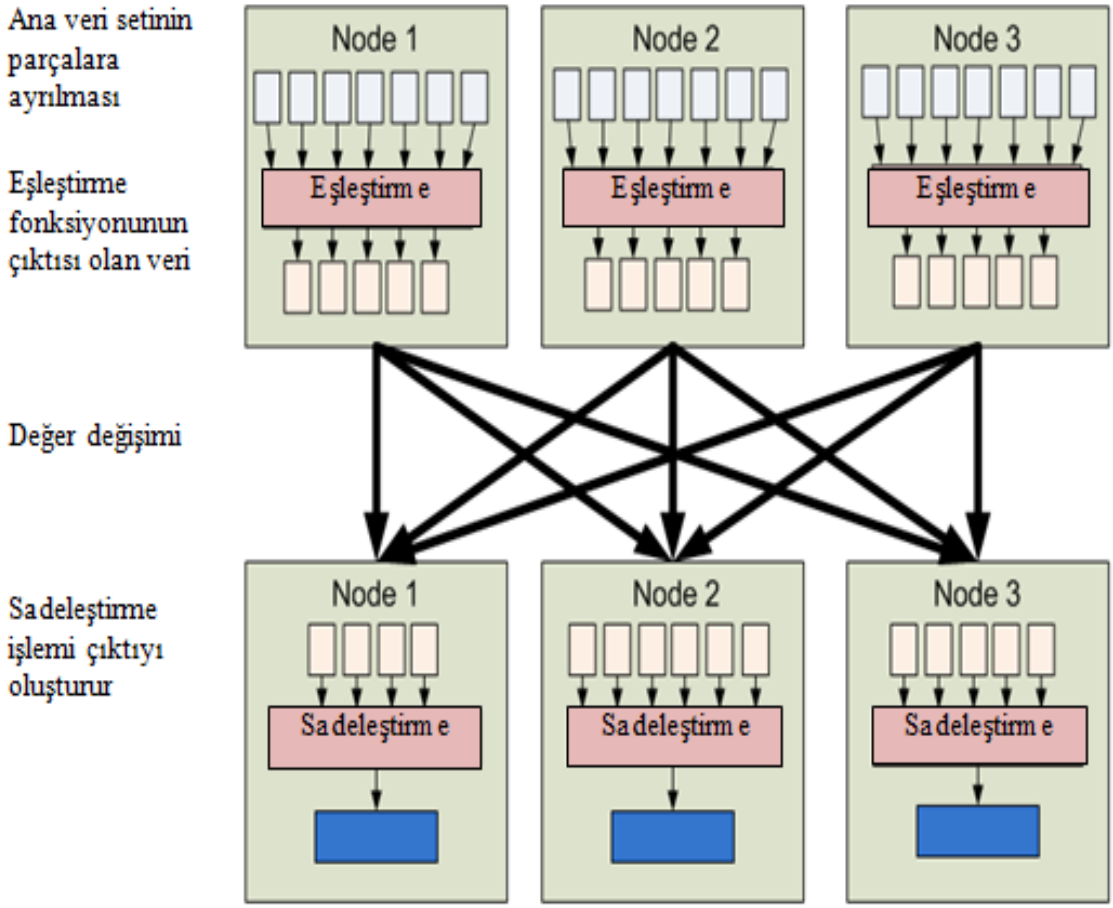


Şekil 2.13 Eşleştirme işlemi [10]

Şekil 2.13'te görüldüğü üzere eşleştirme fonksiyonu ana veri setini parçalara bölmektedir. Eşleştirme işlemi ile ayrılan yığınlar daha sonra "sadeleştirme (reduce)" fonksiyonunun girdisi olmaktadır. Şekil 2.14'te sadeleştirme işlemi görülmektedir. Sadeleştirme işlemi ile ayrılan yığınlar bir bütün haline getirilerek fonksiyonun çıktısını oluşturmaktadır.



Şekil 2.14 Sadeleştirme işlemi [10]



Şekil 2.15 Eşleştirme-sadeleştirme veri akış şeması [10]

Şekil 2.15'te MapReduce paradigmasının çalışması mekanizması gösterilmektedir. Şekildeki değer değişimi birbiri ile ilgili verilerin kümelenmesi işlemini ifade etmektedir. Kümelenen veriler sadeleştirme işlemiyle birleştirilmekte ve sonucu oluşturulmaktadır.

MapReduce paradigmasının güçlü yönlerinden birisi kendiliğinden ölçeklenebilir olmasıdır. 10 gigabayt büyüklüğündeki bir veriyi işlemek için tasarlanan bir program 10 petabayt büyüklüğündeki verileri de işleyebilmektedir. MapReduce girdi verisinin büyüklüğüne bakmadan verileri paralel olarak işler [10].

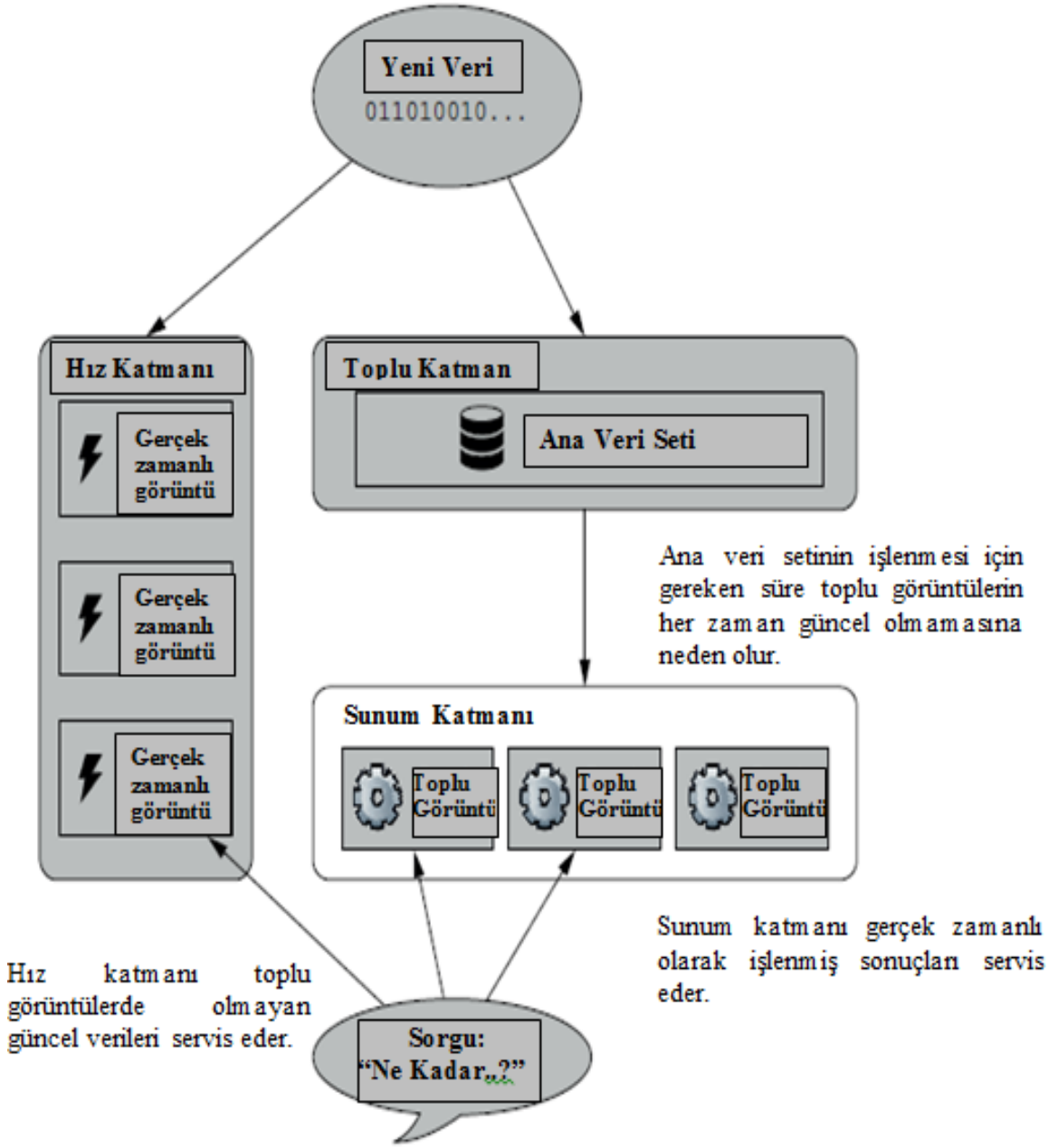
MapReduce paradigmasında eşleştirme ve sadeleştirme işlemlerinin kararlı olması gerekmektedir. Her durumda program aynı sonuçları döndürmelidir. Dahili disklerin kapasitelerine ulaşması, donanımda oluşacak hatalar gibi nedenlerle eşleştirme ve sadeleştirme işlemlerinde veri kaybının olmaması gerekir. Hadoop'un MapReduce ara yüzü gibi ara yüzler bu sorunları hata oluşan düğümlerdeki işlemleri diğer düğümlere

aktararak çözmektedir. Kullanılan MapReduce ara yüzünün hata toleransına sahip olması sonuçların kararlılığı için gereklidir [11].

MapReduce gibi işlem modeli olan bir başka sistem de Spark'tır. Spark MapReduce den daha fazla ölçeklenebilir değildir ama Spark tekrar tekrar aynı veri seti üzerinde işlem yapan algoritmalar için daha yüksek performans sağlamaktadır. Çünkü Spark veriyi diskten her zaman okumak yerine, veriyi hafızada ön belleğe yerleştirmektedir.

2.4. Sunum Katmanı

Sunum katmanı Lambda mimarisinde toplu işlemlerin en son kısmıdır. Toplu katmanı sunum katmanındaki görüntülerin güncellenmesinden sorumlu olduğu için, sunum katmanı toplu katmana bağlıdır. Bu katmanda bulunan görüntüler toplu işlemlerin gecikme süresinin uzun olmasından dolayı her zaman güncel değildir. Ama bu sorun hız katmanı tarafından çözüldüğü için, sistemde sorun oluşturmamaktadır. Şekil 2.16'da Lambda mimarisi ve sunum katmanının işleyişi görülmektedir.



Şekil 2.16 Sunum katmanında veri erişiminde düşük gecikme sağlanması [1].

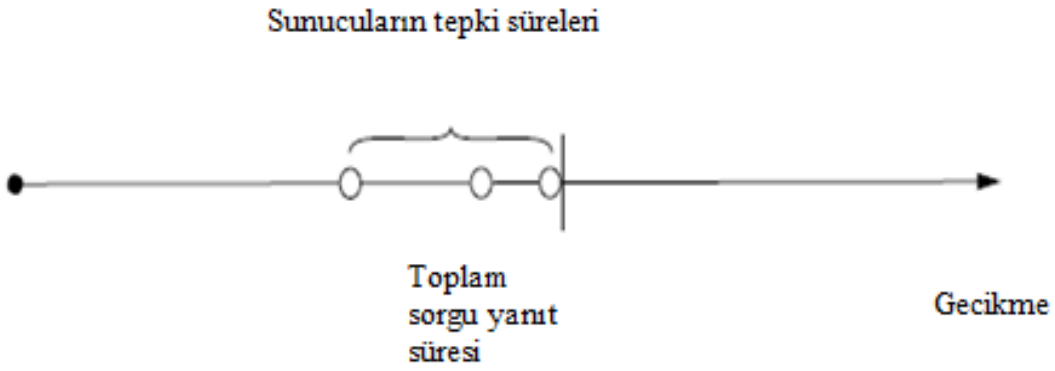
2.4.1. Sunum katmanının performans özellikleri

Sunum katmanı, toplu katmandaki ana veri setinden üretilen toplu görüntülerin servis edildiği katmandır. Sunum katmanında servis edilen veriler ilişkisel formda saklanmadığı için bu katmanda kullanılacak aracın, dağıtık yapıda olması ölçeklendirmenin çok daha kolay yapılmasını sağlayacaktır. Sunum katmanı dağıtık

yapıda tasarlandığında, veri indeksleri tamamıyla dağıtık yapıda oluşturulup servis edilmelidir [1].

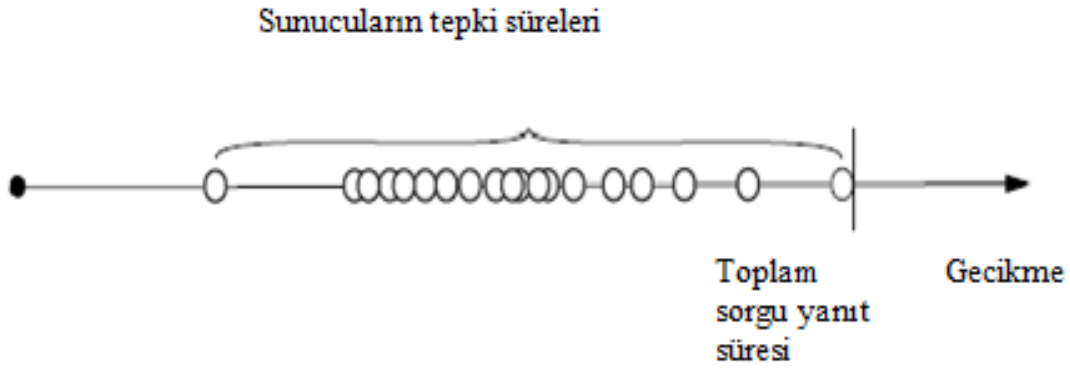
Sunum katmanının indeksleri oluşturulurken performansı etkileyecek iki değişken düşünülmelidir. Bunlar: gecikme ve işlem hacmidir. Gecikme, tek bir sorgunun sonucunun oluşturulması için gerekli süredir. İşlem hacmi ise belirli bir zaman sürecinde servis edilecek sorgu miktarını ifade etmektedir.

Dağıtık mimarilerde, sistemde bulunan makinelerin sayısı arttıkça sorgu yanıt süresinin artma olasılığı da artmaktadır. Bu durumu anlamak için şöyle bir örnek verilebilir: Üç adet sunucunun olduğu bir sistemde, bir sorguya verilecek cevabın süresi, bu üç adet sunucunun o sorguya vereceği cevabın süresine bağlıdır. Sorgunun cevap süresi, en yavaş olan sunucunun hızına bağlıdır. Şekil 2.17'de bu durum gösterilmektedir.



Şekil 2.17 Dağıtık mimaride sorgunun üç sunucuya dağıtılması. [1]

Genel olarak, bir sorgu ne kadar çok sunucuya ulaşırsa, sorgunun gecikme süreside o kadar yüksek olabilmektedir. Bunun nedeni de sistemde sunucu sayısını artırdıkça, en az bir sunucunun geç cevap verme olasılığının artmasıdır. Şekil 2.18'de bu durum görülmektedir.



Şekil 2.18 Sunucu sayısının artmasıyla sorgu sonuçlarının gecikmesi [1].

Sonuç olarak, bir sunucunun en kötü performansı ile sorguların gecikme süresi oldukça fazla olabilmektedir. Bu durum gerçek zamanlı sistemlerde oldukça önemli bir sorun olabilir.

Diğer bir problem de, özellikle sunucularda katıhal diskler (solid-state disk) yerine mekanik sabit diskler kullanıldığında, anahtar değer çözümlerinin işlem hacminin düşük olmasıdır. Bir anahtarın, eşleştirildiği değeri alabilmek için disk üzerinde arama yapılmalıdır. Tek bir sorgu için, birden fazla değerin diskler üzerinden alınması gerekebilir. Disk arama, sıradan sabit diskler için oldukça pahalı işlemlerdir. Sistemde belirli sayıda diskler olacağından, saniyede diskler üzerinde yapılabilecek arama sayısı sınırlıdır. Örnek vermek gerekirse; bir sorgu için ortalama olarak 20 anahtara erişimin gerektiği, bir diskin saniyede ortalama olarak 500 arama yapabildiği ve toplamda 100 adet diskin olduğu bir sistemde, sistemin saniyede cevap verebileceği sorgu sayısı 2.500'tür. Bu rakam disk sayısına göre oldukça az bir değerdir.

Sunum katmanında farklı bir indeksleme stratejisi ile gecikme süreleri ve işlem hacminde iyileştirme yapılabilmektedir. Sunum katmanında bulunan verilerin gruplandırılması gibi ya da normalize edilmemiş formda saklanması gibi işlemler ile performans artırımı yapılabilmektedir [1].

2.4.2. Sunum katmanının normalizasyon / de-normalizasyon sorununa çözümü

Sunum katmanı ilişkisel veri tabanlarının, önemli problemlerinden birisini çözmektedir: Verilerin normalize edilmiş / edilmemiş formda saklanacağına karar verilmesi.

Verilerin hangi formda saklanacağı sistem kurucusunun tercihidir. İlişkisel veri tabanlarında, veriler normalize edilmiş formda saklandığında, birbirinden bağımsız veriler arasında ilişkiler oluşturularak, veri tekrarlarının önüne geçilmektedir. Ama normalize edilmiş bir veri tabanında sorgular çok yavaş çalışabilir. Sorgulara hızlı cevap alabilmek için veriler de-normalize edilmiş formda saklanır. Bu durum performansı artırır ama verinin tutarlılığını sağlamak için büyük bir karmaşa gerekmektedir.

Bu durumu anlamak için şöyle bir örnek verilebilir: Veritabanında kullanıcıların konum bilgilerinin yer aldığı tablolar bulunmaktadır. Verilerin normalize edilmiş formu Çizelge 2.2 ve Çizelge 2.3'te görülmektedir.

Çizelge 2.2. Normalize edilmiş formda kullanıcı bilgilerinin saklandığı tablo

Kullanıcı No	Adı	Soyadı	Konum No
1	Ahmet	Demir	2
2	Ayşe	Alay	1
3	Mehmet	Yılmaz	3

Çizelge 2.3. Konum bilgilerinin saklandığı tablo

Konum No	Şehir	İlçesi	Mahallesi
1	Ankara	Etimesgut	Şehit Osman
2	İstanbul	Şişli	Esentepe
3	Trabzon	Araklı	Bereketli

Çizelge 2.2 ve Çizelge 2.3'te, kullanıcıların konum verilerinin normalize edilmiş hali görülmektedir. Her konum bilgisi kendine ait bir no bilgisine sahiptir. Konum verisinin konum no sütunu ile kullanıcı tablosunun kullanıcı no sütunu birbirleriyle eşleştirilerek tablolar birleştirilmiştir. Bu tablolardan herhangi bir kullanıcının konum bilgisi çekilmek istendiğinde, iki tablo birbirleriyle birleştirilip sorgunun cevabı döndürülecektir. Veri tabanlarında birleştirme işlemleri oldukça maliyetli işlemlerdir. Bu

örnekte kullanıcının konum bilgisi iki tablonun birleşmesiyle elde edilebilmektedir. Tablolar arasında ilişkilendirmelerin daha fazla olduğu veri tabanlarında bir sorgunun sonucu çok daha fazla tablonun birleştirilmesi ile elde edilebilmektedir. Bu durumda sorguların cevap süreleri daha uzun olacaktır.

Normalize edilmiş veri tabanlarındaki performans sorununu aşmak için veriler normalize edilmemiş formda saklanabilmektedir. Çizelge 2.4'te gösterilen örnekte; kullanıcının ad, soy ad gibi bilgileri ile konum bilgileri aynı tabloda tutulmuştur. Bu yüzden sorgular daha hızlı çalışacaktır.

Çizelge 2.4. Normalize edilmemiş formda kullanıcı bilgilerinin saklandığı tablo

Kullanıcı No	Adı	Soyadı	Konum No	Şehir	İlçesi	Mahallesi
1	Ahmet	Demir	2	İstanbul	Şişli	Esentepe
2	Ayşe	Alay	1	Ankara	Etimesgut	Şehit Osman
3	Mehmet	Yılmaz	3	Trabzon	Araklı	Bereketli

Çizelge 2.5. Konum bilgilerinin saklandığı tablo

Konum No	Şehir	İlçesi	Mahallesi
1	Ankara	Etimesgut	Şehit Osman
2	İstanbul	Şişli	Esentepe
3	Trabzon	Araklı	Bereketli

Çizelge 2.4 ve 2.5'te veriler normalize edilmeden, tablolar arasında ilişkiler oluşturulmadan saklanmaktadır. Bu durum sorguların cevap süresini kısaltsa da, verilerin gereksiz yere tekrar tekrar saklanmasına neden olmaktadır. Veri tabanındaki tabloların

ilişkilendirilmeden saklanmasının diğer bir dezavantajı da; verilerin tutarlılığının sağlanmasının çok zor olmasıdır. Verilen örnekte konum tablosundaki verilerdeki değerler değiştiğinde, kullanıcı tablosundaki veriler de güncellenmelidir. Aksi halde veriler arasında tutarsızlık oluşabilmektedir. Bu yüzden verilerin bu formda saklanması ideal bir çözüm değildir.

Lambda mimarisinde ana veri seti ile sunum katmanının ayrılması verilerin hangi formda saklanması gerektiği sorununu ortadan kaldırmaktadır. Ana veri seti, eğer veriler arasında ilişkilendirme fazla ise normalize edilmiş formda saklanmalıdır. Ana veri seti üzerinde ani sorgular yapılmadığından; veritabanı şemasının ani sorgulara göre dizayn edilmesine gerek yoktur. Bu duruma tamamlayıcı olarak, sunum katmanı servis edeceği sorgulara uygun hale getirilebilmektedir. Çizelge 2.6 ve 2.7'de optimize edilmiş sunum katmanı verileri görülmektedir. En yüksek performansa ulaşabilmek için sunum katmanı istenildiği gibi optimize edilebilir. Sunum katmanında yapılan bu optimizasyonlar, verilerin normalize edilmemiş formda saklanmasından çok daha öteye gidebilmektedir [1].

Çizelge 2.6. Sunum katmanında veriler istenildiği gibi optimize edilebilir.

Kullanıcı No	Adı	Soyadı	Konum No	Şehir
1	Ahmet	Demir	2	İstanbul
2	Ayşe	Alay	1	Ankara
3	Mehmet	Yılmaz	3	Trabzon

Çizelge 2.7. Konum tablosu

Konum No	Şehir	İlçesi	Mahallesi
1	Ankara	Etimesgut	Şehit Osman Avcı
2	İstanbul	Şişli	Esentepe
3	Trabzon	Araklı	Bereketli

Lambda mimarisinde verilerin tutarlılığı konusuna gelindiğinde, sunum ve toplu katman arasında veriler gereksiz yere saklanacağı kesinlikle doğrudur. Bu mimarideki anahtar fark ise, sunum katmanın ana veri seti üzerinde çalışan bir fonksiyon olmasıdır. Eğer sunum katmanında tutarsızlık olursa bu hata, ana veri seti üzerinden tekrar işlem yapılarak çözülebilmektedir [1,12].

2.4.3. Sunum katmanı depolama aracı için gerekli olan özellikler

Lambda mimarisine sunum katmanı için kullanılacak veri tabanının belirli özelliklere sahip olması gerekmektedir. Bunlar:

- Toplu görüntülerin yazılabilir olması: Toplu görüntüler ana veri setinden oluşturulmaktadır. Bu yüzden toplu görüntülerin yeni bir versiyonu olduğunda, güncellenen yeni görüntüler eski görüntüler ile yer değiştirebilmelidir.
- Ölçeklendirilebilir olması: Sunum katmanında kullanılan veritabanı herhangi büyüklükteki görüntüleri yönetebilecek kapasitede olmalıdır. Bu yüzden görüntüler dağıtık yapıdaki makinelere dağıtılmalıdır.
- Rastgele okuyabilme: Sunum katmanı veritabanı görüntülerin küçük kısımlarına direk erişimini sağlayacak indeksleme ile birlikte rastgele okumalara izin vermelidir.
- Hata toleransı: Sunum katmanı veritabanı dağıtık yapıda olduğu için makine hatalarını tolere edebilmelidir. Bir makine çöktüğünde o makinede bulunan verilere erişim sağlanabilmelidir.

Sunum katmanı için gerekli olan özellikler arasında olmayan ama bütün veri tabanlarının standartlarından biri olan özellik rastgele yazım özelliğidir. Toplu görüntüler sunum katmanından bağımsız oluşturulduğu ve ana veri seti üzerinde yapılan değişiklikler için toplu görüntüler güncellenmesi gerektiği için bu katmanda rastgele yazma işlemlerine ihtiyaç yoktur. Lambda mimarisinde rastgele yazma işlemleri anlık sorgulara hızlı cevap verebilme özelliği hız katmanında ele alınmaktadır [1].

Rastgele yazma; işlemleri veri tabanlarındaki karmaşıklığın en önemli nedenlerinden biri olduğu için, sunum katmanında bu özelliğin aranmıyor olması bu

katmanın yapısını basite indirgemektedir. Bu katmanda rastgele yazımlarım yapılmaması veri tabanı üzerinde sıkıştırma işlemlerinin gerekliliğini de ortadan kaldırmaktadır. Sunum katmanı ana veri setinin çok büyük bir bölümünü içeren görüntüleri barındırdığından, ana veri seti kaynağı için gerekli olan kaynağın büyük bir kısmı bu katman için de gereklidir. Sıkıştırılmış ana veri setinin kaynakları kadar bir kaynak da sunum katmanına ayrılacağından bu katmanda sıkıştırma işlemler ile uğraşılmamaktadır [1,6].

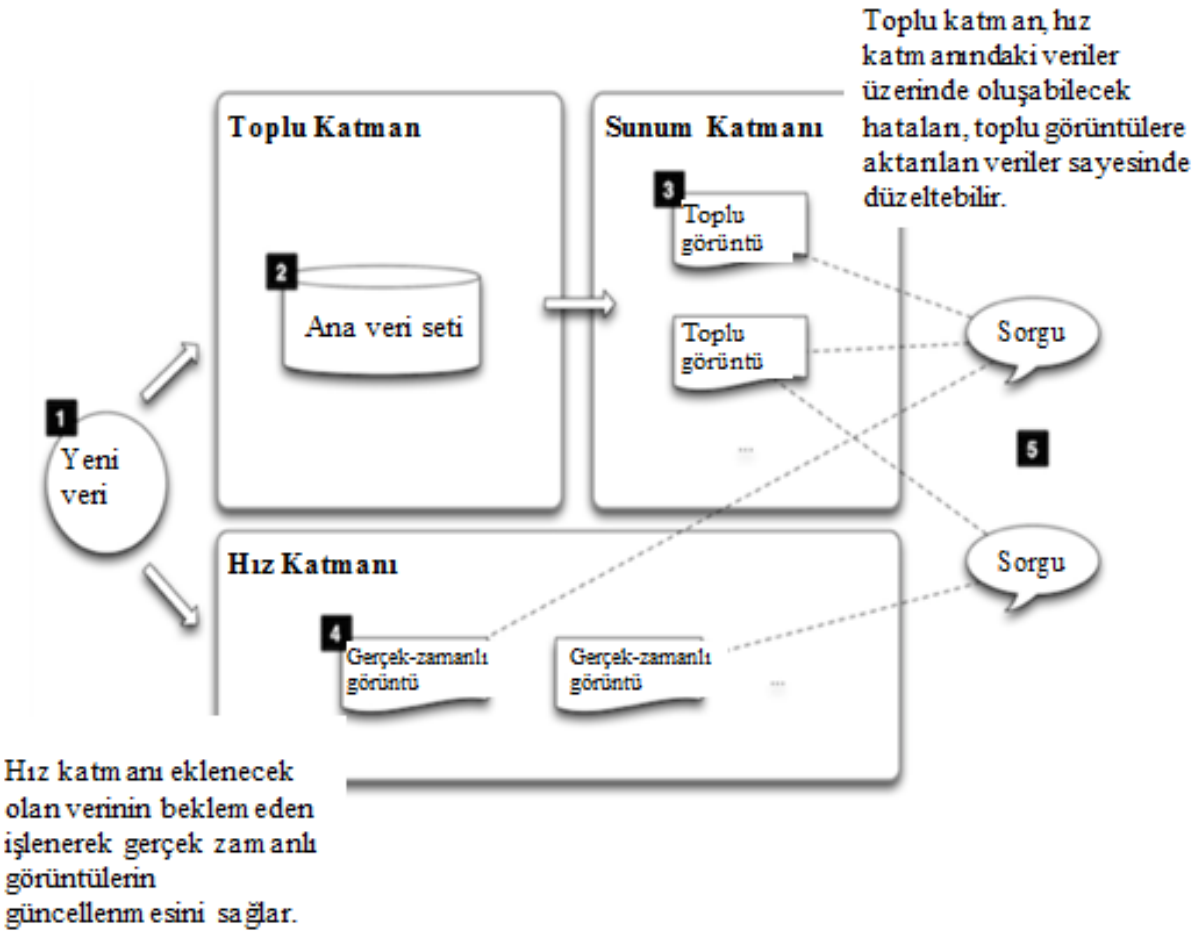
Veri tabanlarının rastgele yazımları desteklediği durumlarda, veri tabanlarının uğraşması gereken diğer bir zorluk; yazma ve okuma işlemlerinin senkron olarak yapılmasıdır. Yazma ve okuma işlemleri senkron olarak çalışmadığında eksik aktarılmış veriler okunabilmektedir. Bir veritabanı, yazma işlemlerini desteklemezse daha iyi bir performans elde edebilmektedir.

Sunum katmanının genel kavramlarını özetlemek gerekirse aşağıdaki maddeler sıralanabilir:

- Gecikme ve işlem hacminde iyileştirme yapabilmek için görüntüler şekillendirebilmektedir.
- Rastgele yazımları desteklemediği için katmanın basitliği sağlanmıştır.
- Toplu katmanda normalize edilmiş veri saklanırken, hız katmanında verinin normalize edilmemiş hali saklanabilmektedir.
- Ana veri seti tekrar işlenerek sunum katmanının verilerini güncelleyeceği için kendiliğinden hata toleransına ve veri düzeltilmesine sahiptir.

2.5. Hız Katmanı

Lambda mimarisinin son katmanı olan hız katmanı bir veri sisteminde olması gereken en önemli özelliklerden biri olan veri tabanına rastgele yazma işlemlerinin gerçekleştirildiği katmandır.



Şekil 2.19 Lambda mimarisinde hız katmanı [15]

Şekil 2.19'da Lambda mimarisinde yeni gelen verinin işlenerek gerçek zamanlı görüntüleri oluşturması görülmektedir. Gerçek zamanlı görüntüler, güncel verilerden üretilen ve sunum katmanının sorgulara cevap vermesine yardım eden görüntülerdir.

Tüm ana veri seti üzerinde toplu işlem yapmak hem zaman hem de kaynak maliyeti anlamına gelmektedir. Mümkün olduğunca güncelleme işlemlerinin hızlı yapılabilmesi için hız katmanı, toplu katman ve sunum katmanından daha farklı bir strateji ile oluşturulmalıdır. Bu yüzden hız katmanı toplu işlem algoritmasıyla değil, artımlı algoritma üzerine kurulmalıdır. Artımlı algoritma, gelen her yeni verinin işlemden geçirilerek sonuç oluşturulduğu algoritmadır [1,6].

Artırımlı algoritma, toplu işlemlerden oldukça karmaşıktır. Ama hız katmanının sorumlu olduğu veri, en fazla birkaç saatlik ve ana veri setinden çok daha küçük olduğundan bu karmaşıklık azalmaktadır. Hız katmanındaki görüntülerin geçici olması da bu katmanın karmaşasını azaltmaktadır. Hız katmanında saklanan veriler, ana veri seti

üzerinden tekrar toplu görüntüleri oluşturulduğunda silinebilmektedir. Hız katmanı; daha kompleks bir yapıya sahip ve bu katmanda hata oluşma olasılığı daha yüksek olsa da, hatalar kısa sürelidir. Çünkü toplu ve sunum katmanları ile bu hatalar düzeltilebilmektedir.

Lambda mimarisinin gücü, tüm görevleri katmanlara ayırmaktan gelmektedir. İlişkisel veritabanları ile kurulan geleneksel veri sistemleri, tüm işlemler tümüyle hız katmanı gibi davranmaktadır. Bu yüzden artımlı algoritmaların karmaşasını çözmek için sınırlı seçenekleri vardır [1].

Hız katmanının iki ana görevi vardır. Bunlar:

- Gerçek zamanlı görüntüleri saklamak
- Gelen veriyi işleyerek bu görüntüleri güncellemektir.

2.5.1. Gerçek zamanlı görüntüleri oluşturmak

Hız katmanında oluşturulan görüntülerin amacı, toplu görüntülerde olduğu gibi verileri etkili bir şekilde sorgulayabilmektir. Aralarındaki en önemli fark, gerçek zamanlı görüntülerin; yeni veri geldiğinde, kısa zaman içerisinde güncellenmesi gerekliliğidir. Zamanın kısalığı uygulamadan uygulamaya değişebilir ama genel olarak sorgu cevap sürelerinin birkaç mili saniyeden bir kaç saniye aralığında değişmesi beklenmektedir [1].

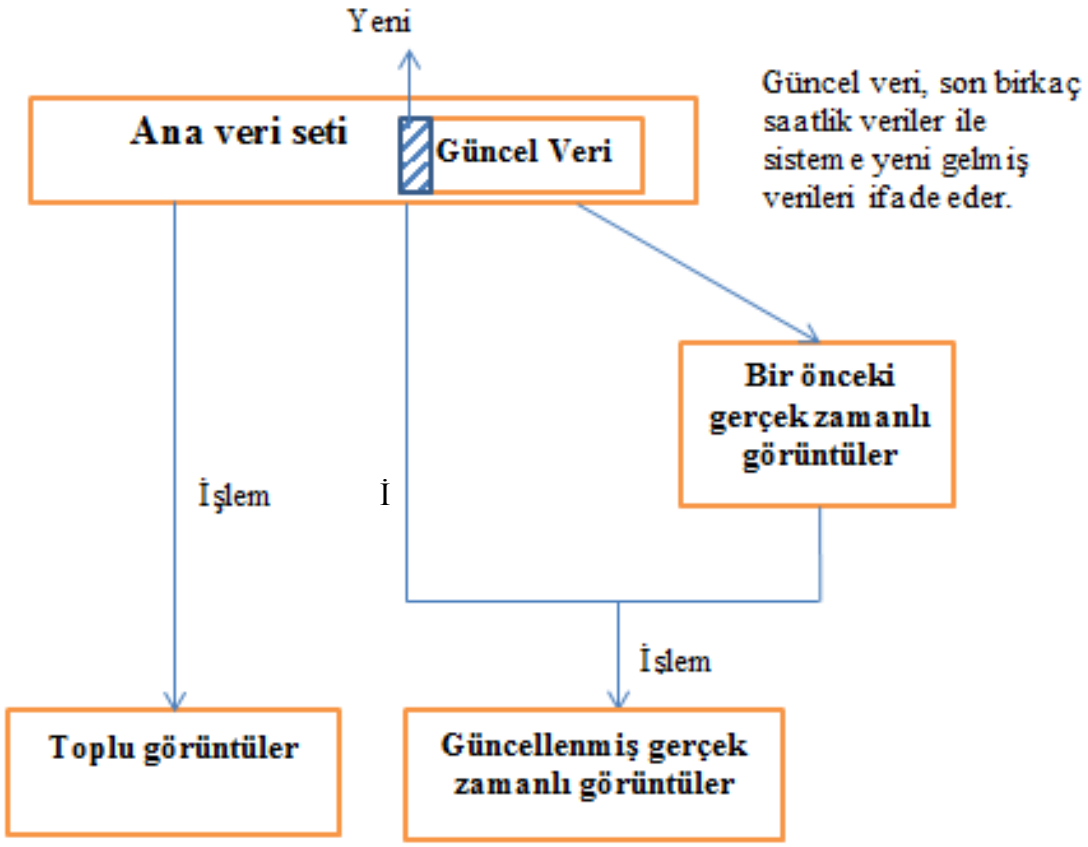
İstenilen veri sisteminde anlık olarak yeni veriler görüntülenmek isteniyorsa, gerçek zamanlı görüntüler milli saniyelik gecikmeler ile oluşturulmalıdır. Hız katmanının genel çalışma stratejisi aşağıdaki eşitliktir:

gerçek zamanlı görüntüler = işlem (güncel veri)

Bu çalışma stratejisi, toplu görüntülerin çalışma stratejisi ile aynıdır. Tek fark toplu görüntülerin tüm ana veri seti, gerçek zamanlı görüntülerin ise güncel veri üzerinde işlem yapmasıdır.

Şekil 2.20 Güncel verilerden gerçek zamanlı görüntülerin oluşturulması

Şekil 2.20'de Lambda mimarisinde oluşturulan görüntülerin kaynakları görülmektedir.



Şekil 2.21 Yeni veri ulaştığında gerçek zamanlı görüntülerin güncellenmesi

Şekil 2.21'de görüldüğü gibi yeni veri geldiğinde gerçek zamanlı görüntüleri güncelleyebilmek için daha önce üretilen görüntüler kullanılmaktadır.

2.5.2. Gerçek zamanlı görüntüleri saklamak

Hız katmanı için kullanılacak veritabanı diğer katmanlarda kullanılan veri araçlarından daha özellikli olmalıdır. Bu katmanda kullanılacak olan veri aracı aşağıdaki belirtilen özellikleri sağlamalıdır:

- Rastgele okuma: Gerçek zamanlı bir görüntü düşük gecikme ile hızlı okuma yapabilmelidir.
- Rastgele yazma: Artırımlı algoritmanın sağlıklı çalışabilmesi için çok düşük gecikmeler ile görüntüler güncellenebilmelidir.
- Ölçeklendirme: Sunum katmanından olduğu gibi bu katmanda sakladığı verinin büyüklüğüne ve sistemin talep ettiği okuma/ yazma oranına göre ölçeklendirme yapabilmelidir. Bu durum genellikle gerçek zamanlı görüntülerin birden fazla makineye dağıtılması anlamına gelmektedir.
- Hata toleransı: Makinelerden ya da disklerden birisi çökerse, katman normal bir şekilde çalışmaya devam edebilmelidir. Bazı depolama araçlarında verinin kopyası birden fazla makinede tutularak, bir makinede hata olduğunda diğer makinelerdeki yedekler ile bu sorun aşılabilmektedir.

Hız katmanı için gerekli olan özellikler birçok NoSQL veritabanı tarafından sağlanmaktadır. NoSQL veritabanları birçok veri modeli ve indeks türünü desteklemektedirler. Bu yüzden MongoDB, Cassandra, CouchDB gibi NoSQL veri tabanlarından bir tanesi hız katmanı için kullanılabilir.

Hız katmanında saklanan verinin büyüklüğü

Bu katmanda bulunan görüntüler yalnızca güncel verileri içerdiği için, saklanan veri miktarı ana veri setine göre çok küçüktür. Bu katmanda bulunan görüntüler, sunum katmanında bulunan görüntülerden çok daha karmaşık olduğu için, verinin azlığı sistemi idame ettirenlerin yararınadır [1].

Sunum katmanının karşılaşmadığı ama bu katmanda ortaya çıkan zorluklar aşağıda sıralanmıştır:

- Çevrimiçi sıkıştırma: Veritabanında güncelleştirmeler olduğunda, disklerin bazı kısımları kullanılamaz olmaktadır. Veritabanı, bu alanları tekrardan kullanılabilir hale getirebilmek için periyodik olarak sıkıştırma işlemi yapacaktır. Sıkıştırma işlemi oldukça fazla kaynak tükettiğinden, sıkıştırma sırasında sorgulara hızlı cevap vermesi gereken kaynakların önemli bir kısmı kullanılacaktır. Sıkıştırma işlemi düzgün bir şekilde yapılmadığında tüm sistemin çökmesine neden olacak hatalar zincirini başlatabilmektedir.

- Uyumluluk(concurrency): Okuma ve yazma işlemlerinin yapıldığı veritabanları, aynı değer için aynı anda hem okuma hem de yazma işlemleri ile karşı karşıya kalabilmektedir. Veritabanının, atılan sorgulara tutarsız ya da eski değerler döndürmemesi için yazma ve okuma işlemleri koordine edilmelidir. Kilitleme gibi kontrol stratejileri bu koordineyi sağlasa da, hataya oldukça meyillidirler.

Lambda mimarisinde her katmanın kendi görevinin olması, hız katmanının karmaşasını azaltmaktadır. Eğer bu katman tamamen artırımlı algoritma ile çalışırsa, oluşacak veri yoğunluğu azaltılmış olmaktadır. Daha küçük veri setleriyle uğraşmak daha kolaydır [1].

2.5.3. Artırımlı algoritmanın zorlukları

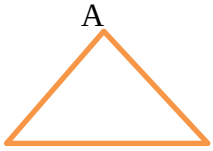
Artırımlı algoritmaların, yeniden işleme algoritmalarına göre insan hata toleransı daha düşüktür. Ama artırımlı algoritmalar daha performanslı çalışmaktadırlar [1].

Hız katmanında ilk öncelik hızdır. Verilerin hızlı bir şekilde işlenip görüntülerin güncellenmesi gerekmektedir. Artırımlı algoritmalar ile dağıtık yapıda bir sistem kurulurken, CAP teoremi ile artırımlı algoritmalar arasındaki etkileşimi anlayabilmek önemlidir.

CAP Teoremi 1998 yılında Eric Brewer tarafından ortaya atılan, özellikle dağıtık mimariye sahip bir sistemin aynı anda, kullanılabilirlik (availability), tutarlılık (consistency) ve bölünme toleransı (partition tolerance) özelliklerini aynı anda bulunduramayacağını söylemektedir. Bu teoreme Brewer Teoremi de denir[16].

Kullanılabilirlik (Availability)

A



C P

Tutarlılık (Consistency) Bölünme Toleransı (Partition Tolerance)

Şekil 2.22 CAP Teoremi

CAP teoremi makineler arasında iletişim bozukluğu olduğunda, sistemin ya kullanılabilir ya da tutarlı olabileceğini; bu iki özelliğin aynı anda sağlanamayacağını söylemektedir. Dağıtık mimaride çalışan veri araçları, Şekil 2.22'de gösterilen CAP üçgenin CP ya da AP kenarında yer almaktadır. Tutarlılık, her düğümün aynı anda aynı veriyi görmesidir. Bir sistemin tutarlı olabilmesi için, tutarlı bir halde başlayan hareketin, tutarlı bir halde sonlanması gerekmektedir. Hareket sırasında, bir hata oluşursa o hareketteki tüm işlemler iptal edilir. Bu yüzden hareket sırasında veriler tutarsız hale gelirse, tüm işlemler iptal olacağından sistemin tutarlılığı tekrar sağlanmaktadır.

Kullanılabilirlik, sistemin her daim isteklere cevap döndürmesidir. Sistem, mevcut düğümlerin durumlarına bakılmaksızın, her kullanıcıya bir cevap döndürmesidir.

Bölünme toleransı, ağ hatalarından dolayı oluşacak ani bölünmelere rağmen sistemin çalışmaya devam etmesidir. Bölünme toleransı olan sistemler, tüm ağ hatasına neden olmayacak herhangi büyüklükteki bir ağ hatası oluştuğunda, çalışmaya devam edebilmektedirler.

CAP teoremine göre dağıtık mimarili bir sistemin her sorguya her zaman en güncel veriyi döndürebilmesi için, düğümlerin her zaman birbirine sağlıklı bir ağ ile bağlı olması gerekmektedir. Dağıtık mimariye sahip sistemlerin dezavantajlarından birisi ağların her zaman sağlıklı çalışmamasıdır. Herhangi bir ağ sorunu olduğunda ne yapılacağının belirlenmesi gerekir. Bu durumda iki seçenek mevcuttur: sistemin tutarlı olması ya da kullanılabilir olmasıdır [17].

Tutarlılık ve Bölünme Toleransı (CP– Consistency / Partition Tolerance)

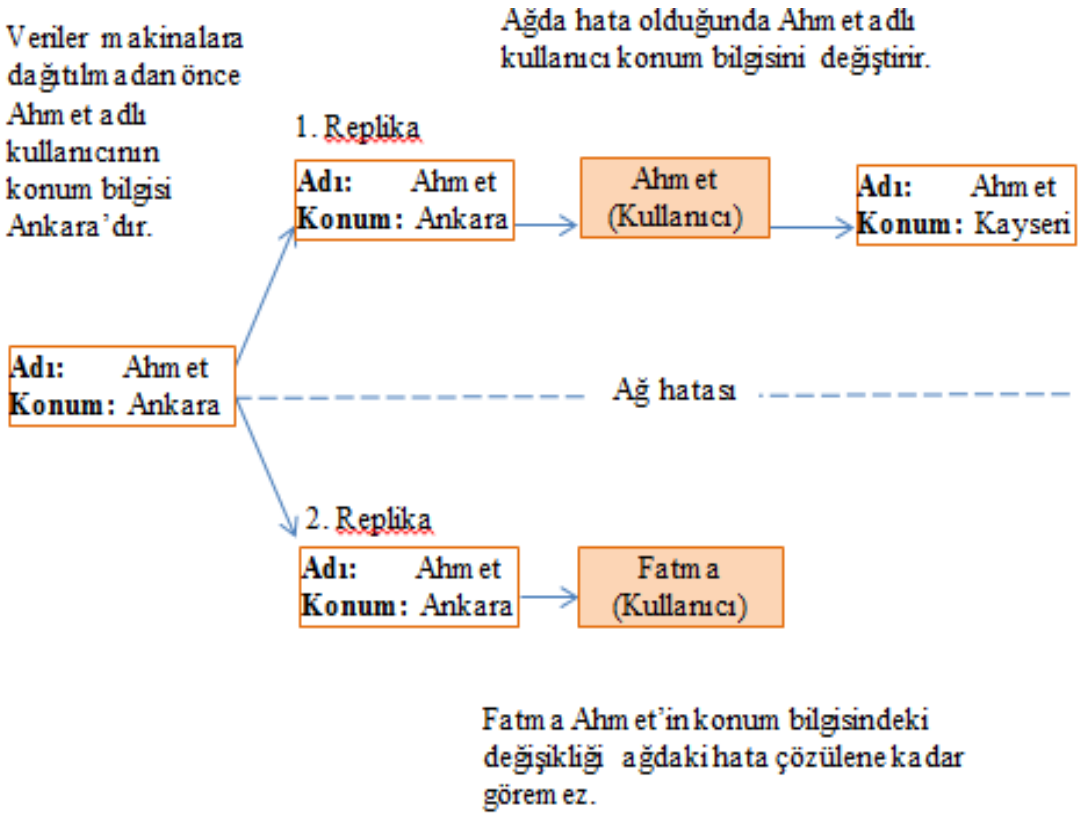
Ağ hatası olduğunda sistemin hata toleransını artırabilmek için, bazı veri sistemleri aynı veriyi birden fazla sunucu üzerinde kopyalayarak saklamaktadırlar. Böylelikle bir

replikaya erişim sağlanamadığında diğer sunuculardaki replikalara erişim sağlanabilmektedir. Tüm replikalara erişimin sağlanamaması durumu, daha düşük olasılıkla olsa da mümkündür [17].

Asıl önemli soru, yazma işlemi olduğunda sistemin nasıl davranacağıdır. Sistem, yazma işlemi olduğunda bu işlemleri reddedebilmektedir. Bu yöntem ile her okuma talebi için verinin en son hali döndürülmektedir. Bu durumda, her kullanıcı aynı değeri göreceğinden sistemin tutarlılığı sağlanmış olmaktadır. Tasarlanan senaryoya göre sistem hata da dönebilmektedir.

Kullanılabilirlik ve Bölünme Toleransı (AP– Availability / Partition Tolerance)

Sistemde ağ hatası olduğunda, izlenebilecek diğer senaryo da, erişilebilir olan replikaların yazma işlemini kabul etmesidir. Erişim sağlanamayan replikalara erişim sağlandığında veriler senkronize edilmektedir.



Şekil 2.23 Ağ hatası sırasında replikaların farklılaşması

Şekil 2.23'teki örnekte ağ hatası sırasında yazma işlemine izin verildiği için birbirinden farklılaşan replikalar görülmektedir. Bu örnekte farklı makinaların alt grupları ile iletişim halinde oluşan iki kullanıcı bulunmaktadır. Ahmet adlı kullanıcı ağda hata

olduğunda konum bilgisini değiştirmiştir. Ahmet yeni konum bilgisini görebilirken, Fatma adlı kullanıcı Ahmet adlı kullanıcının yeni konum bilgisini ağdaki hata düzeltildikten sonra görebilmektedir.

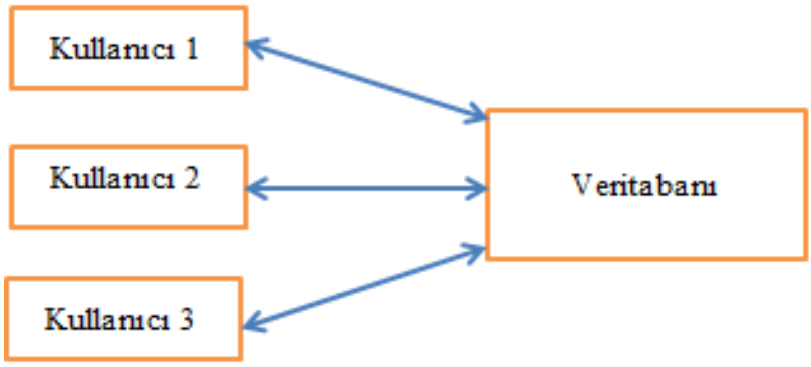
Ağ hataları sırasında yazma işlemine izin verildiğinde replikların birbirinden farklılaşması kaçınılmaz olmaktadır. Bu stratejide, sistem ulaşılabilir olsa da, tutarlı değildir. Bu örnek ağda hata oluştuğunda, sistemin hem erişilebilir hem de tutarlı olmasının mümkün olmadığını göstermektedir [17].

Lambda mimarisinde kullanılacak araçların geneli dağıtık mimariye sahip araçlardır. Bu yüzden mimarideki katmanlar, CAP teoremi altında değerlendirilmelidirler. Toplu katmanındaki yazma işlemleri, ana veri setine eklenecek yeni gelen verilerdir. Bu katmandaki her veri parçası birbirinden bağımsız olduğundan, yazma işlemleri sırasında makinelerin koordine olarak çalışmasına ihtiyaç duyulmaz. Eğer gelen veri veritabanına yazılamaz ise; veri arabelleğe alınıp, daha sonra tekrar denenebilmektedir. Sunum katmanında ise; toplu görüntüler yüksek gecikme ile oluşturulduğundan, okunan veriler genellikle eski verilerdir. Dolayısıyla her iki katman doğası gereğince CAP teoreminin erişilebilirlik ve bölünme toleransı alanında kalmaktadır.

2.5.4. Asenkron ve senkron güncelleme

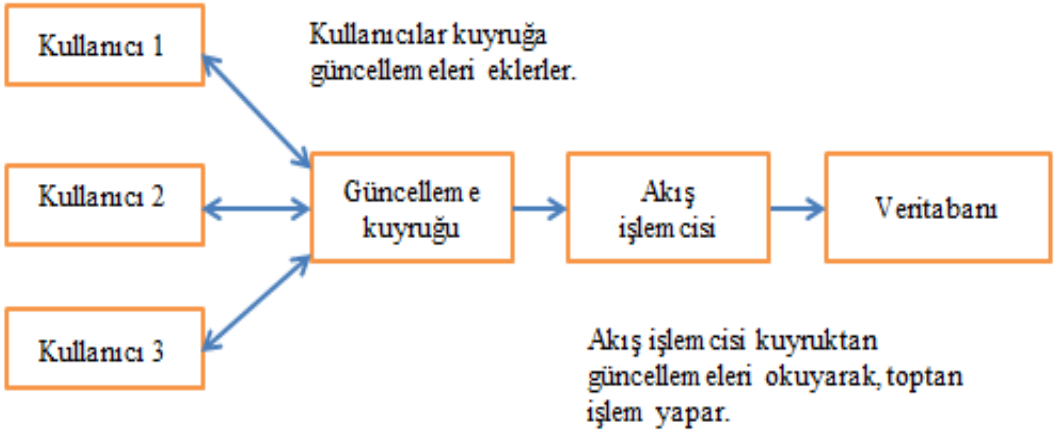
Hız katmanın mimarisi, gerçek zamanlı görüntülerin senkron ya da asenkron olarak güncellenmesine göre değişmektedir.

Senkron güncelleme, uygulamanın direk olarak veritabanına istek gönderdiği ve güncelleme bitene kadar veritabanının engellendiği güncellemelerdir. Senkron güncellemeler veritabanına direk bağlandıkları için hızlıdırlar. Ayrıca güncelleme ile uygulamanın koordine olmasına olanak sağlamaktadır. Örnek vermek gerekirse; kullanıcı güncellemenin bitmesini beklerken, uygulama üzerinde uyarı gösterilebilmektedir. Senkron güncelleme için hız katmanın mimarisi Şekil 2.24'te gösterilmektedir.



Şekil 2.24 Senkron güncellemeler kullanan basit bir speed katmanı

Senkron güncellemelerin tersine, asenkron güncellemeler talepleri bir kuyruğa ekleyerek, işlemleri daha sonra yapmaktadır. Hız katmanında gecikme birkaç mili saniye ile birkaç saniye arasında olsa da, taleplerin çok fazla olması bu süreyi uzatabilmektedir. Asenkron güncellemeler, senkron güncellemelere göre daha yavaş çalışır. Çünkü veritabanı değiştirilmeden önce, güncellemeler ek işlemlerden geçmektedir. Asenkron güncellemeler, işlem sırasında kontrol edilemediğinden, başka eylemler ile koordine edilemezler. Ama asenkron güncellemeler birçok avantaja sahiptir. Öncelikle kuyruktan birden fazla mesaj okunabilmekte ve veritabanına toplu güncellemeleri yapılabilmektedir. Bu da işlem hacmini oldukça artırmaktadır. Ayrıca değişen yükü de rahatlıkla idare edebilirler. Eğer talep sayısı artarsa, güncellemeler bitene kadar kuyruk ek talepleri saklar. Diğer taraftan senkron güncellemelerde talep trafiği, veritabanının aşırı yüklenmesine neden olabilmektedir. Bu durum da taleplerin düşmesine ve zaman aşımı gibi hataların oluşmasına yol açabilir.



Şekil 2.25 Asenkron güncelleme mimarisi

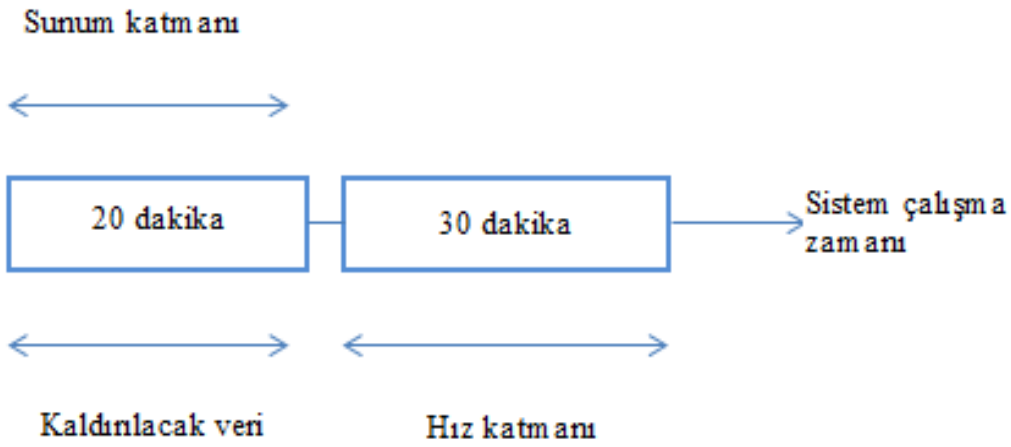
Asenkron güncellemeler için hız katmanının mimarisi Şekil 2.25'te gösterilmektedir. Senkron güncellemeler kullanıcılar ve kullanıcı arayüzü ile etkileşimi olan sistemlerde kullanılmaktadır. Asenkron güncellemeler ise, analiz tabanlı ve kullanıcılar ile etkileşim gerekmeyen sistemlerde kullanılmaktadır. Asenkron güncellemelerin avantajları, daha iyi iş yüküne sahip olmaları ve yük fazlalığı olduğunda daha kolay idare edilebilirliktir.

2.5.5. Gerçek zamanlı görüntülerin silinmesi / sonlandırılması

Artırımlı algoritmalar ve rastgele yazımı destekleyen veritabanları, hız katmanını sunum ve toplu katmanlardan çok daha fazla karmaşık yapmaktadır. Ama hız katmanındaki verilerin kısa süreli olması, bu karmaşıklığı azaltmaktadır. Toplu ile sunum katmanı, sürekli hız katmanındaki verileri geçersizleştirdiğinden; hız katmanının yalnızca toplu ve sunum katmanında yer almayan verileri barındırması gerekmektedir. Toplu katmanı işlemlerini bitirdiğinde, toplu görüntülerde yer alan veriler hız katmanından silinebilir.

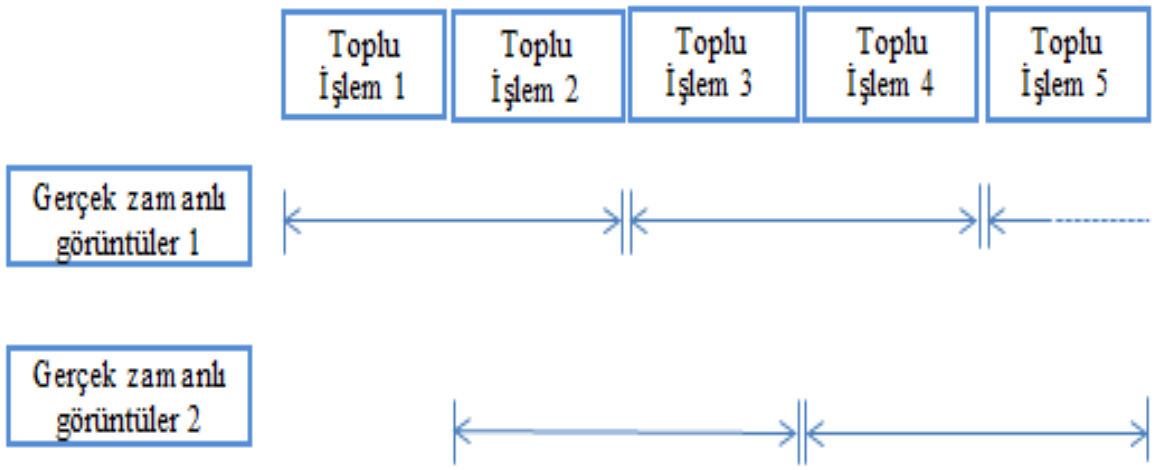
Katmandaki görüntüler kaldırılırken, veritabanın çalışmasını etkilememelidir. Bu yüzden tam olarak hangi görüntülerin kaldırılacağının bilinmesi gerekir. Lambda mimarisi uygulanıp ilk kez sistem çalıştırıldığında, toplu ve sunum katmanında veri olmayacaktır. Toplu katman ilk kez çalıştığında, boş veri seti üzerinde işlem yapacaktır. Örneğin; toplu katmanının işlem süresini yirmi dakika olduğu bir sistemde, ilk yirmi dakikanın sonunda sunum katmanındaki görüntülerde hiçbir veri olmayacaktır.

Toplu katmanı ikinci kez çalıştığında, ilk yirmi dakikada sisteme girilen verileri işlemeye başlayacaktır. Bu işlemin otuz dakika aldığı düşünülürse, işlem bittiğinde sunum katmanındaki veriler ilk yirmi dakikada işlenen veriler olacaktır. Bu durumda hız katmanına ilk yirmi dakikada aktarılan veriler silinmelidir. Bu durum Şekil 2.26'da görülmektedir.



Şekil 2.26 Gerçek zamanlı görüntülerden kaldırılacak veri

Toplu katmanı üçüncü kez çalıştığında, sunum katmanındaki görüntülerde ilk elli dakikalık veri yer alacaktır. Hız katmanındaki ilk otuz dakikalık veri fazlalık olacaktır. Bu veriyi içeren gerçek zamanlı görüntüler Hız katmanından kaldırılmalıdır. Bu işi yapabilmek için en kolay yol gerçek zamanlı görüntüleri iki kopya olarak tutmak ve toplu katmanı her çalıştığında bu kopyaları sıra ile temizlemektir. Şekil 2.27'de bu algoritma görülmektedir.



Şekil 2.27 Gerçek zamanlı görüntülerin iki kopyasının bulunması

Şekil 2.27'de gösterilen gerçek zamanlı görüntülerden biri her zaman sunum katmanında olmayan verilere sahip olacaktır. Toplu katman her çalıştığında, uygulama daha çok veri olan görüntü setinden okumaya başlamalıdır.

3. İL-CAS VE LAMBDA MİMARİSİ

İL-CAS Bankamız arşiv verilenin sistemli bir şekilde saklanmasını, merkez ve taşra teşkilatını kapsayacak şekilde, Banka tarafından üretilen/kullanılan klasik (çıktı veya pafta) ve sayısal coğrafi verilerin sayısal ortamda arşivlenmesi ile beraber kurumun harita üretim projelerinden gelen sayısal verilerin sistematik bir biçimde kaydedilebilmesini, kurum içinde yapılmakta olan sayısal harita ve plan üretim çalışmalarının ISO-TC211 ve OGC tarafından tanımlanmış olan veri standartlarına uygun bir şekilde depolanmasını, Banka coğrafi veri envanterinin web tabanlı uygulamalar ile kurum içi ve kurum dışı kullanıcılara ISO ve OGC standartlarına tam uyumlu olarak kullanılabilmesini ve aynı zamanda diğer kurumlardan yayınlanan/yayınlanacak coğrafi veri servislerinin çevrim içi kullanılabilmesini sağlayan otomasyon sistemidir.

İL-CAS'ta hedeflenen kullanıcıların başında Banka personelleri ve belediyeler gelmektedir. Ayrıca İL-CAS verilerinin diğer kamu kurumlar ile web servisler ile sunulması da hedeflenmektedir.

3.1. İL-CAS Verileri

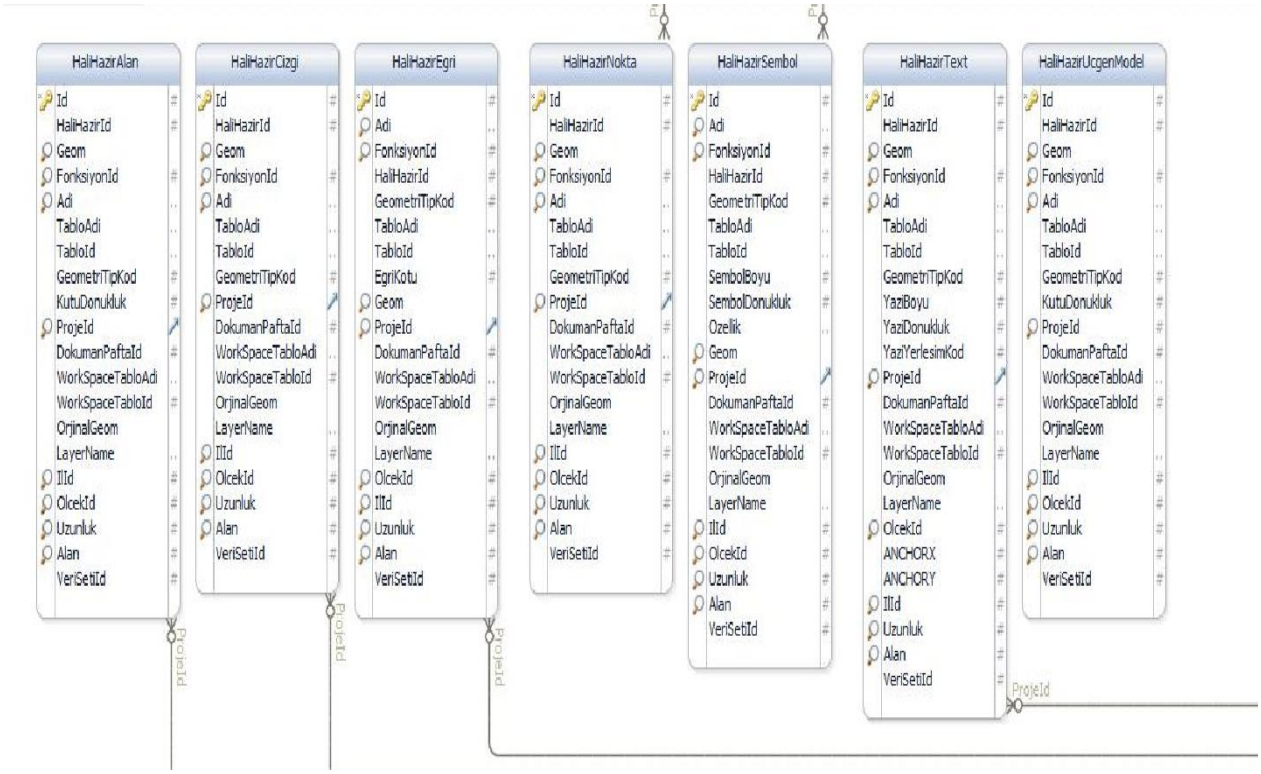
İL-CAS'ta veritabanı olarak PostgreSQL kullanılmaktadır. Kullanılan versiyon 9.4'tür. PostgreSQL, tek bir sunucu üzerinde çalışan, çoklu istemcilere cevap verebilen açık kaynak kodlu ilişkisel bir veritabanı sistemidir [18].

3.1.1. Sayısal veriler

Sayısal veriler veritabanında vektör veri olarak saklanan verilerdir. Vektör veriler; konum bilgisine sahip, nokta, çizgi ve alan olmak üzere üç farklı geometriyle ifade edilen veri tipidir. Sayısal verilerin PostgreSQL veritabanında saklanabilmesi için Postgis eklentisi kullanılmaktadır. Postgis, coğrafi verilerin koordinatları ile birlikte veritabanı üzerinde tutulmasını sağlayan ve bu verileri işleyebilecek fonksiyonlara sahip olan açık kaynak kodlu bir eklentidir.[19,20].

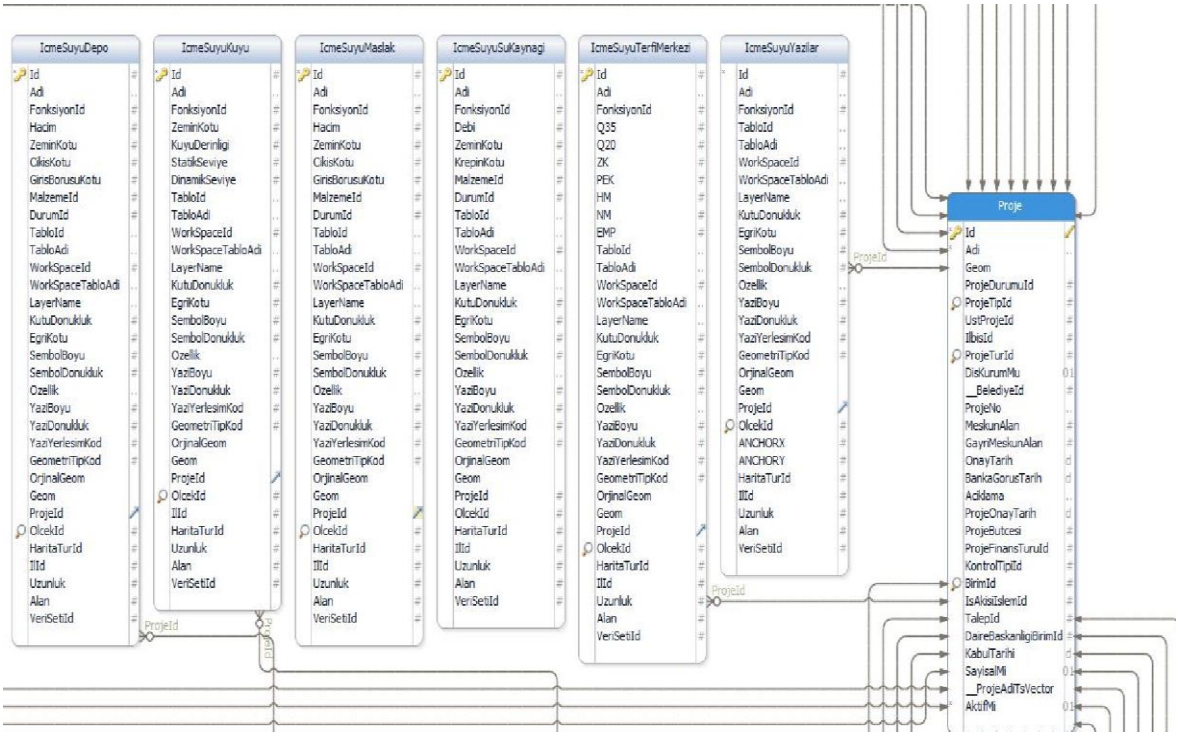
Sayısal veriler içerisindeki her bir vektör bilgisine nesne denebilir. PostgreSQL üzerinde bulunan vektör verilerin büyüklüğü yaklaşık olarak 6 terabayttır. Her bir nesne veritabanı coğrafi veri tablolarındaki bir sütuna tekabül etmektedir. Örnek vermek gerekirse, sayısal bir harita verisinde 200 000 nesne varsa, veritabanında o sayısal harita

verisine ait coğrafi tablolarda 200 000 kayıt bulunduğu anlamına gelir. Her bir coğrafi nesne tipi için ayrı tablolar oluşturulmuştur. Şekil 3.1'de sayısal haritalar için oluşturulan tablolardan bir kesit gösterilmektedir.



Şekil 3.1 Her bir nesne türünün farklı tablolarda saklanması.

Tek bir projeye ait binlerce kayıt bulunduğu için bu tablolar da il değerlerine göre bölünmüşlerdir. Veritabanında veriler, proje bazlı saklanmaktadır. Bu yüzden her bir coğrafi nesne tablosu ana proje tablosu ile ilişkilendirilmiştir. Her projenin ili, ilçesi, belediye bilgileri bulunmaktadır. Şekil 3.2'de Postgres üzerinde tabloların kısmi bir görüntüsünde, coğrafi tabloların proje tablosu ile ilişkilendirildiği görülmektedir.

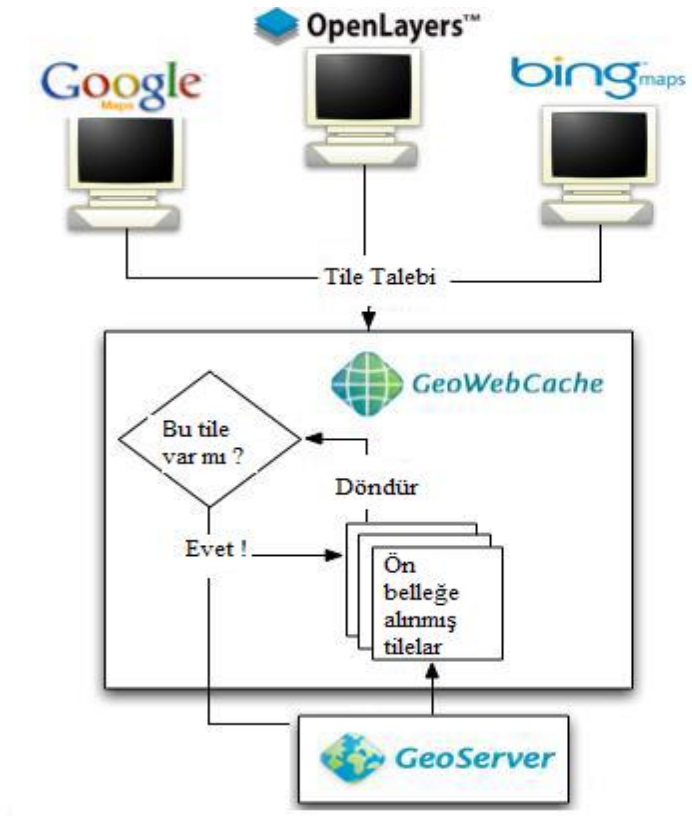


Şekil 3.2 Coğrafi tabloların proje tablosu ile ilişkilendirilmesi.

İL-CAS projesinde coğrafi veri sunucu olarak GeoServer kullanılmaktadır. GeoServer, OGC standartlarında verilerin servis edilebilmesini sağlayan açık kaynak kodlu bir uygulamadır. GeoServer, WMS ve WFS standartları ile verilerin servis edilmesini sağlar [21].

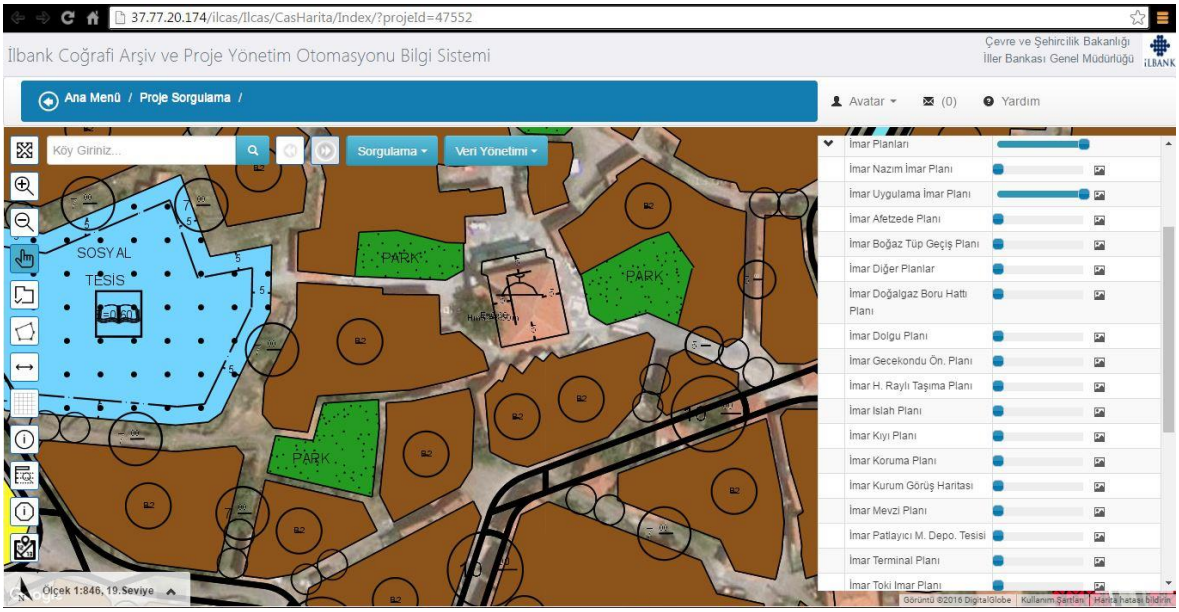
Sayısal veriler, İL-CAS üzerinde iki ayrı mekanizma ile görüntülenmektedir. Bunlar: statik ve dinamik katman mekanizmalarıdır.

Statik katmanda veriler GeoServer'in GeoMapCache mekanizması kullanılarak ön belleğe alınmaktadır. GeoServer eğer ön belleğe alma stratejisine uygun düşen hazır bir nesne varsa GeoServer işlemlerine gerek kalmadan tile katman olarak ön belleğe edilmiş nesneyi istemciye servis eder. Şekil 3.3'te GeoMapCache'in nasıl çalıştığı görülmektedir.



Şekil 3.3 GeoMapCache çalışma mekanizması [21]

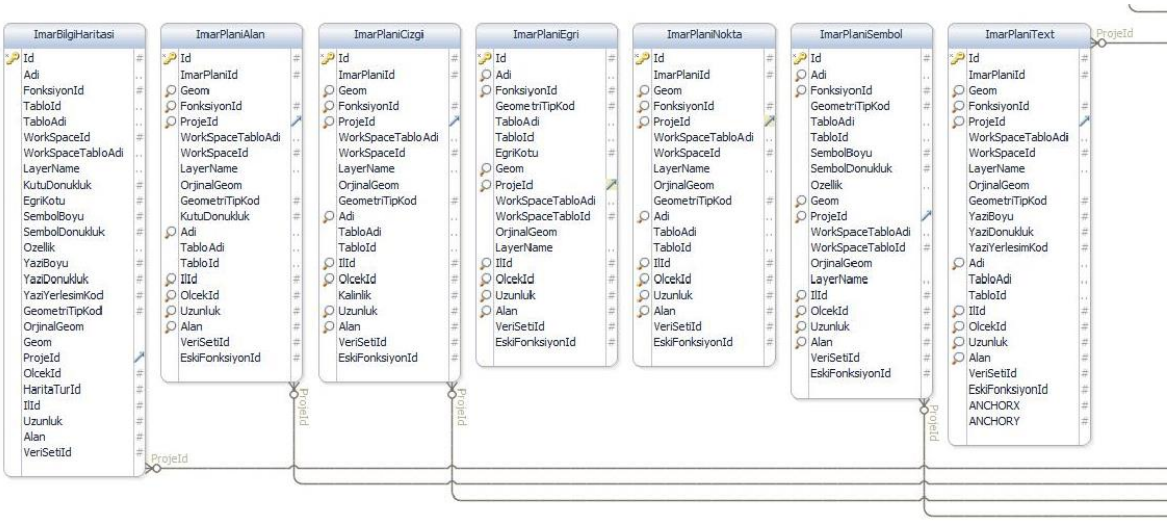
Projede on dokuz tane ölçek seviyesi bulunmaktadır. Ölçek seviyesi sayısal verilerin hangi yakınlıkta ya da uzaklıkta görüntüleneceğinin belirlenmesini sağlamaktadır. GeoMapCache; her bir katmanın verisini, her bir ölçek seviyesi için, veritabanından çekmektedir. Çalışan uygulamadan bağımsız olarak, nesneleri harita üzerinde çizerek, her bir resim dosyasının köşe noktaları koordinatlandırmaktadır. Köşeleri koordinatlandırılmış her bir resim dosyasına tile denir. Tile dosyalarının her biri 256*256 piksel büyüklüğündedir.



Resim 3.1 İL-CAS katman yapısı.

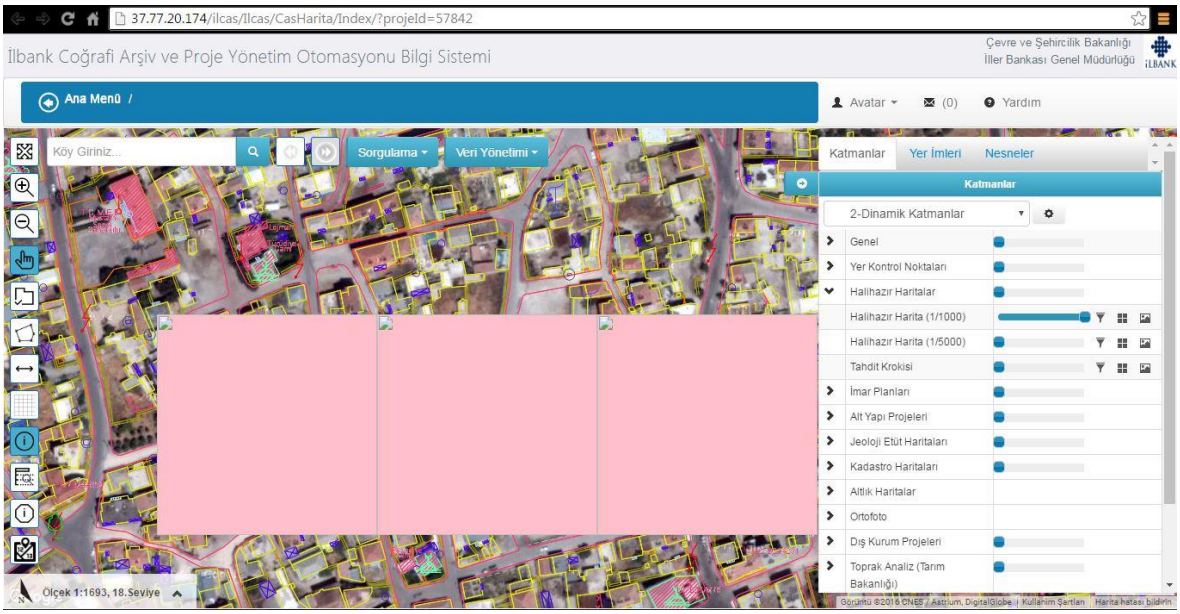
İL-CAS'taki ön belleğe alma stratejisi, tile katmanları oluşturma üzerine kurulmuştur. Sayısal verilerin PostgreSQL'den çekilerek her bir ölçek için, her bir katmanın tileları oluşturulmaktadır. Buradaki katman kavramı veritabanındaki proje türüne tekabül etmektedir. Resim 3.1'de imar planlarına ait katmanlar görülmektedir. Nazım imar planı, uygulama imar planı, afetzede imar planı gibi katmanlar imar planının türünü ifade eder. Bunlardan her biri birer katmandır. Şekil 3.1'de görülen coğrafi tablolarda yer alan "layername" özelliği o nesnenin hangi katmana ait olduğunu göstermektedir

Resim 3.1'deki katmanlardan biri olan nazım imar planı katmanı üzerinden örnekleme yapılırsa; veritabanından Şekil 3.4'te görülen imar planlarına ait her bir nesne tablosundan, "layername" özelliği nazım imar planı olan tüm kayıtlar çekilir. GeoMapCache mekanizmasıyla, birden on dokuza her katman için veritabanından çekilen nesneler GeoServer tarafından çizilip koordinatlandırılarak tile olarak kaydedilir. İşlem sonucunda her bir katmanın tile verileri ilgili tile katmanında saklanır.



Şekil 3.4 İmar planlarına ait nesnelerin saklandığı tablolar

Statik katman, sayısal verilerin çok daha hızlı görüntülenebilmesi için oluşturulmaktadır. Statik katmanın dezavantajı verilerin çoğu zaman güncel olmamasıdır. Çünkü ön belleğe alma işlemi oldukça uzun sürmekte ve kaynak tüketmektedir. Bu yüzden ön belleğe alma işlemi yapılırken, sayısal veriler görüntülenmek istendiğinde GeoServer tileları doğru olarak sunamayabilir. Görüntülenemeyen tilelar harita üzerinde pembe kareler şeklinde görülmektedir. Resim 3.2'de pembe tile görüntüsü görülmektedir.



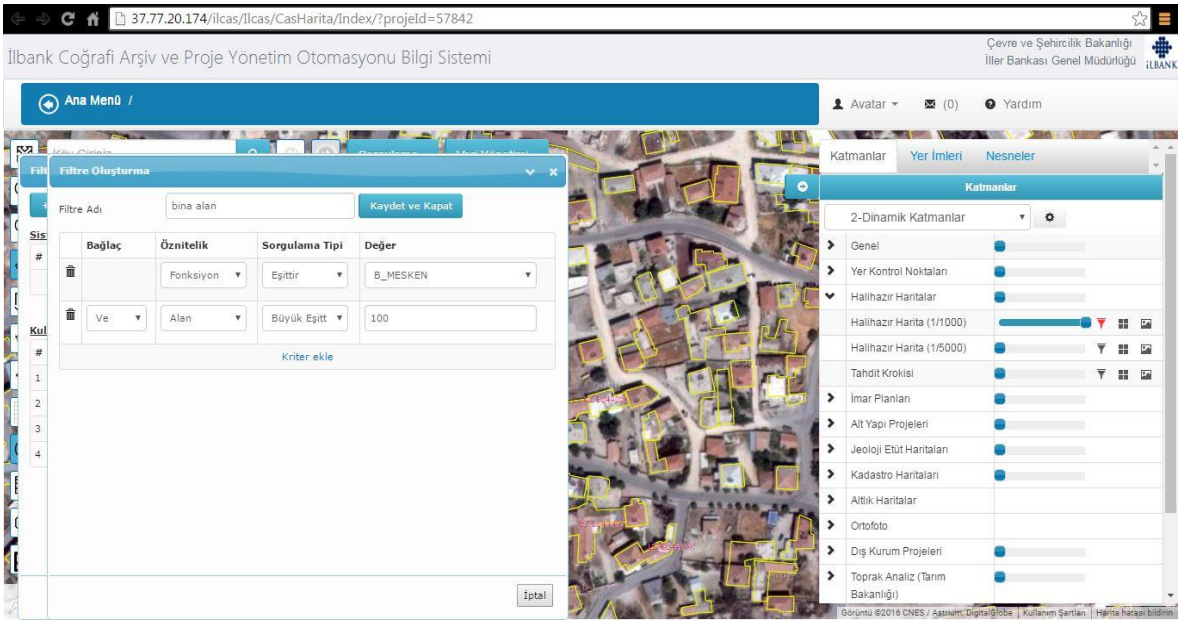
Resim 3.2 Geoserver'ın tileları sunamaması

Ön belleğe alma işlemi oldukça maliyetli bir işlemdir. Ön belleğe alma sırasında, sunucu kaynağının önemli bir kısmı kullanılmaktadır. Ön belleğe alınacak veri fazla olduğu için de, işlem oldukça uzun sürmektedir. Bu yüzden statik katmandan çoğu zaman

güncel verilere erişim sağlanamayacaktır. İL-CAS projesinin hedeflerinden biri de yerinde kontrol yapılmasına gerek olmayan işlerde harita üzerinden kontrolün yapılmasıdır. Veriler, dinamik katman mekanizması ile görüntülenirse çok yavaş olacaktır. Statik katman mekanizması ile görüntülenirse de güncel veriler, o anki statik katman verileri arasında olmayacaktır.

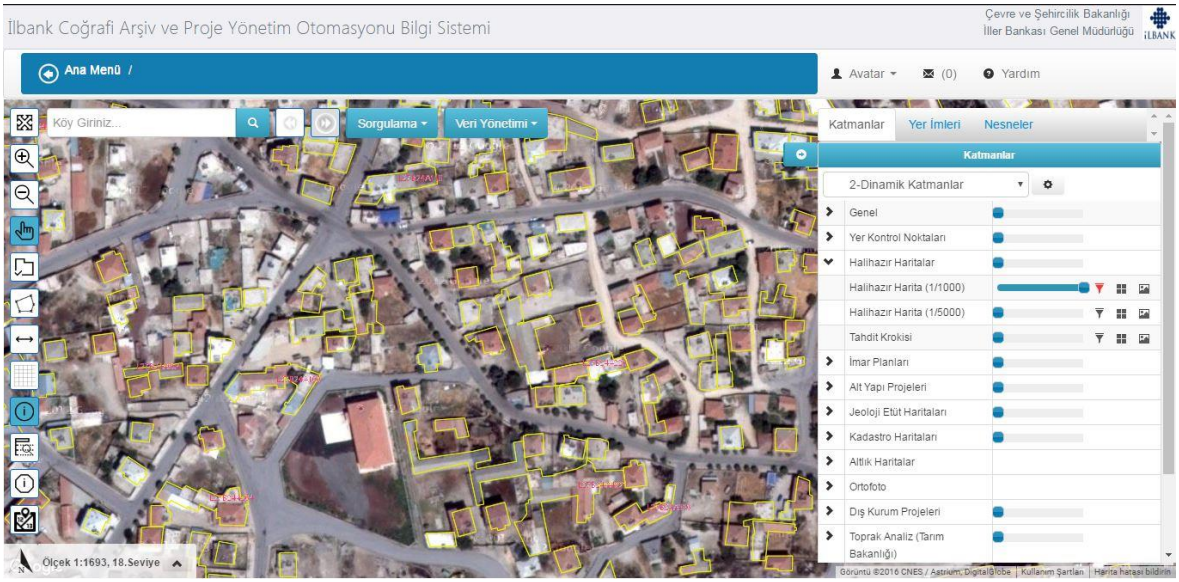
Dinamik katmanda ise, GeoServer tarafından veriler PostgreSQL üzerinden anlık olarak çekilerek, her bir vektör verinin yani her bir nesnenin çizimi sunucu üzerinde yapılmaktadır. Görüntülenmek istenen projede ne kadar nesne var ise, o kadar talep kullanıcı tarafından GeoServer'a, GeoServer tarafından da veritabanına yapılacaktır. Nesne sayısı ne kadar fazla olursa, istenilen verinin görüntülenmesi o kadar yavaş olacaktır.

Dinamik katman ile veriler üzerinde ayrıntılı sorgulamalar yapılabilir. Çünkü veriler vektör halinde görülmektedir. Örnek vermek gerekirse, bir haritanın içinde alanı 100 m^2 'den daha büyük olan mesken alanlar harita üzerinden sorgulanabilir. Resim 3.3 ve 3.4'te bu örneğin ekran görüntüsü görülmektedir.



Resim 3.3 Alanı 100 m^2 'den daha büyük olan meskenlerin sorgulanması

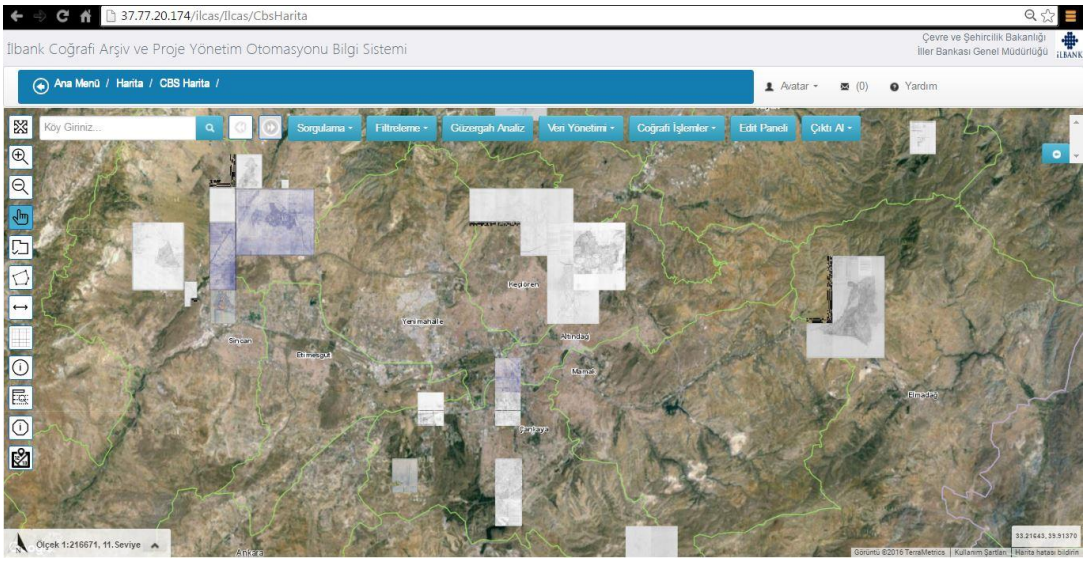
Resim 3.4'te gösterilen sonuç, mesken veri katmanındaki ilgili meskenlerin filtrelenmesinin sonucudur.



Resim 3.4 Alanı 100 m^2 ‘den daha büyük olan meskenlerin sorgu sonucu

3.1.2. Raster veriler

Raster veriler piksellerle ifade edilen mekânsal verilerdir. En genel tanımıyla, koordinatlandırılmış imaj dosyaları olarak düşünülebilir. GeoServer, WCS (Web Coverage Service) ile raster verilerin sunucudan servis edilmesini sağlar. İL-CAS projesinde raster veriler, A3 formatındaki paftaların taranması ve koordinatlandırılması ile oluşturulmuştur. Resim 3.5’te taranmış ve akıllandırılmış paftaların sistem üzerinde gösterildiği bir örnek görülmektedir. İL-CAS’ta taranmış ve akıllandırılmış raster verilerin büyüklüğü 10 terabayt civarındadır [20].



Resim 3.5 Taranmış ve akıllandırılmış paftalar

3.1.3. Diğer veriler

İL-CAS verilerinin bir kısmını da projelere ait, parasal, metin, zaman gibi değerler ile, pdf ve word dosyaları oluşturmaktadır. Word ve pdf dosyalarının sistem üzerinden ön izlemesi yapılabilmekte ve indirilmesi sağlanmaktadır. Bu verilerin büyüklüğü yaklaşık olarak 2 terabayttır.

3.2. İL-CAS ve Lambda Mimarisi

3.2.1. Neden Lambda mimarisi kullanılmalı?

İL-CAS, Bankamızın kurulduğu 1938'den bugüne tüm verilerin arşivlenmesini ve hizmete açılacağı andan itibaren, güncel veriler ile beslenerek ülkemizin mekansal verilerinin önemli bir bölümünün depolanabileceği büyük bir projedir.

Coğrafi veriler, doğası itibariyle yoğun verilerdir. Sayısal ortamda harita üzerinde bir proje görebilmek için veritabanına binlerce kayıt atmak gerekmektedir. Halihazırdaki İL-CAS mimarisinde, geliş ve gidişlerdeki veri yoğunluğunu azaltabilmek için statik ve dinamik katman algoritmaları kullanılmştır. Bu algoritmalara göre statik katmanda veriler ön belleğe alınarak imaj formatında saklanarak, sayısal verilerin daha hızlı görüntülenmesi sağlanmaya çalışılmıştır. Dinamik katman ise, sayısal verileri vektör veri olarak veritabanından direk çekmekte ve harita üzerinde anlık çizmektedir. Bu da dinamik katmanın oldukça yavaş çalışması anlamındadır.

Statik katmandan veriler daha hızlı gelse de, GeoMapCache'in çalıştığı zamanlarda tile katmanlarından veriler hızlı bir şekilde çekilememektedir. Bu yüzden ön belleğe alma işleminin oluşturulduğu katman ile, sorgulamanın yapıldığı katmanın birbirinden ayrılması sistemde okuma işlemlerinin daha hızlı yapılabilmesine olanak sağlayacaktır.

Sisteme rastgele yazma işlemi yapıldığında, tek bir kullanıcı için bu işlem veritabanına binlerce kayıtın yapılması anlamına gelir. Ortalama bir sayısal projede 150 000 nesne var ise ve aynı anda sisteme 100 kişi veri girişi yapmaya çalışsa, veritabanına 1 500 000 kayıt yapılacağı anlamına gelir. Dağıtık mimaride olmayan bir sistemde aynı anda hem cachelemenin yapılması, hem de 1 500 000 kaydın yapılması, hem de sorgulara hızlı bir şekilde cevap verilebilmesi için oldukça yüksek donanım maliyeti gerekir. İL-CAS devreye alındıktan sonra, kurumlar arasında web servisler ile veri paylaşımı yapılırsa, bu

durum erişim sayısını artıracağından sistem daha da yavaş çalışacaktır. Dağıtık mimaride çalışan tek bir NoSQL veritabanının kullanılması da çözüm sağlayamayacaktır. Verileri okuma işleme, veritabanına yazma işlemi ya da verilerin ön belleğe alınma işinin tek bir araç tarafından yapılması, sistemin yavaş çalışmasına, yük fazlalığı olduğunda da sistemin cevap vermemesine hatta sistemin çökmesine bile neden olabilir.

Lambda mimarisi gerçek zamanlı sistemlerde, anlık yazma ve okuma işlemlerinin hızlı yapılabilmesi ve kaynakların verimli kullanılabilmesi için tasarlanmıştır. Mimari, problemi üç katmana dağıtarak, her tür veri sistemine uygulanabilecek genel bir çözüm sunmaktadır [22].

Lambda mimarisi Twitter, Amazon Web Services gibi anlık olarak çok büyük veriler ile uğraşan şirketlerin kullandığı bir mimaridir. Twitter, Hadoop, Cassandra gibi araçları kullanarak tasarlanan mimaride günlük beş milyar oturum ile gelen veriyi işleyebilmektedir [23,24].

Lambda mimarisinin sağladığı düşük gecikmenin tek bir platform ile sağlanabilmesi için çok yüksek maliyetli donanım maliyetlerine ihtiyaç vardır. Örneğin; Google, harita hizmetlerini tek bir NoSQL veritabanı BigTable ile sunmaktadır. Ama haritalar üzerinde sayısal veriler olmamasına rağmen, sorgulara hızlı cevap verilebilmesi için bütün verilerini SSD diskler üzerinde saklamaktadır [25].

İL-CAS projesinde yaşanan sorunlar anlık olarak gelen verinin büyüklüğü, gelen verilerin işlenerek gerçek zamanlı olarak servis edilmesi için gereken işlem fazlalığı ve sayısal veriler üzerinde ön belleğe alma işlemi yapılırken sistemin işlem yoğunluğundan dolayı cevap verememe ihtimalidir. Lambda mimarisi İL-CAS'ta yaşanacak sorunları farklı katmanlara ayırarak çözmektedir. Anlık gelen verileri saklama ve işleme sorununu hız katmanına atmaktadır. Sayısal verilerin ön belleğe alma işlemi ile toplu katman ilgilenirken, verilerin servisi ile de servis katmanı ilgilenmektedir. Sorunlar farklı katmanlara ayrılarak, anlık yoğun verilerin işlenmesini ve mevcut verilerin de hızlı bir şekilde servis edilmesini sağlamaktadır.

3.2.2. İL-CAS için gerçek tabanlı veri modelinin uygulanabilirliği

İL-CAS verilerinin büyük bir bölümünü sayısal veriler oluşturmaktadır. Her bir sayısal veri bütünü bir proje başlığıyla saklanmaktadır.

Verilerin gerçek tabanlı saklanması, yani her bir özelliğin tarihsel sürecinin tutulması özellikle coğrafi veriler için oldukça uygundur. İL-CAS üzerinde sayısal veriler üzerinde değiştirme işlemi yapılmamaktadır. Yeni veriler, yeni proje başlığıyla sisteme atılmaktadır. Harita üzerinde eklendiği tarihe göre en son veri görülmektedir.

Gerçek tabanlı veri modeli verilerin sonsuza kadar doğru olmasını sağlayan bir modeldir. Bu özellik coğrafi verilerin doğasında bulunmaktadır. Örnek vermek gerekirse; bir imar projesinde daha önceden konut olarak planlanmış bir alanın daha sonradan okul alanı olarak değiştirilmesi, o alanın daha önce konu için ayrıldığı gerçeğini değiştirmemektedir. Aynı örnekleme Banka'daki tüm coğrafi verileri için yapılabilir. Banka'nın sahip olduğu verilerin değişmez veri modeli ile depolanması; ülkemizin altyapı, üstyapı ve mekansal verilerinin tarihsel sürecinin saklanmasını sağlar.

3.2.3. Lambda mimarisi için önerilen platformlar

Lambda mimarisi bütün alanlara uygulanabilecek, hata toleransına sahip, ölçeklendirilebilen gelişmiş bir veri algoritmasıdır.

Lambda mimarisi üç katmandan oluşmaktadır. Bunlar toplu, sunum ve hız katmanlarıdır. Toplu katman ana veri setinin saklandığı ve ana veri setinin toplu işlemlerden geçirildiği katmandır. Hız katmanı, iki toplu işlem arasında sisteme girilmiş olan verileri sunacak olan katmandır. Yani gerçek zamanlı verinin saklandığı ve gerçek zamanlı verinin toplu katmandaki toplu işlem algoritmasından geçirildiği katmandır. Sunum katmanı ise hız ve toplu katmanda bulunan, toplu işlemlerden geçmiş indekslenmiş görüntülerin sorgulandığı katmandır. Yani kullanıcının sorgularına cevap veren katmandır.

İL-CAS projesinde Lambda mimarisini uygulamak için seçilecek platformlarda en çok dikkat edilmesi gereken husus, ana veri setini oluşturan verilerin büyük bir kısmının coğrafi veri olduğudur. Seçilecek platformlar, coğrafi verilerin sunulmasından sorumlu olan GeoServer ile entegre olarak çalışabilmelidirler.

Batch Katmanı

İL-CAS ana veri setinde tablolar arasında oldukça yoğun ilişkilendirme yapılmıştır. Bu yüzden ilişkisel olmayan bir veritabanı seçildiğinde, tablolar arasındaki ilişkiler için diyagram şemaları oluşturmak gerekir. Yani her bir ilişkinin kodlanması gerekir. Tablolar arası ilişkilendirilmelerin böyle yoğun olduğu bir ana veri setinde

diyagram şemaları kullanmak insan gücü bakımından oldukça büyük bir maliyet anlamına gelmektedir. Çünkü her bir ilişkinin teker teker kodlanması gerekir. Bu yüzden ana veri setinin ilişkisel veritabanında saklanması mantıklı olacaktır. Bu veri seti üzerinde sorgulama yapılmadığı için verilerin normalize edilmemiş formda saklanmasının bir önemi yoktur. Çünkü rastgele sorgulara hızlı cevap vermek gibi bir yükümlülüğü yoktur. Aslında verilerin normalize edilmemiş formda saklanması daha az depolama kaynağını gerektirmesi de bir avantajıdır. Ayrıca ilişkisel veri tabanlarının transactionları desteklemesi verilerin doğruluğu için önemli bir özelliktir. Bu nedenlerden dolayı toplu katmanda PostgreSQL kullanılması doğru karar olacaktır. PostgreSQL coğrafi verilerin başarılı bir şekilde depolayabilen esnek bir veritabanıdır [18].

Toplu katmanın diğer bir görevi toplu görüntülerin oluşturulmasıdır. İL-CAS projesinde toplu görüntülerin oluşturulma işi, sayısal verilerin tile katmanlarının oluşturulması, pdf/ word dosyalarının daha kolay sorgulanabilir hale getirilmesi ve projelere ait tanımlı veri tiplerinin normalize edilmemiş hale getirilmesidir. Sayısal verilerin tile katmanlarının oluşturulması GeoServer tarafından yapıldığı için toplu görüntülerin oluşturduğu her katmanda seçilecek araç, GeoServer ile entegre olarak çalışmalıdır.

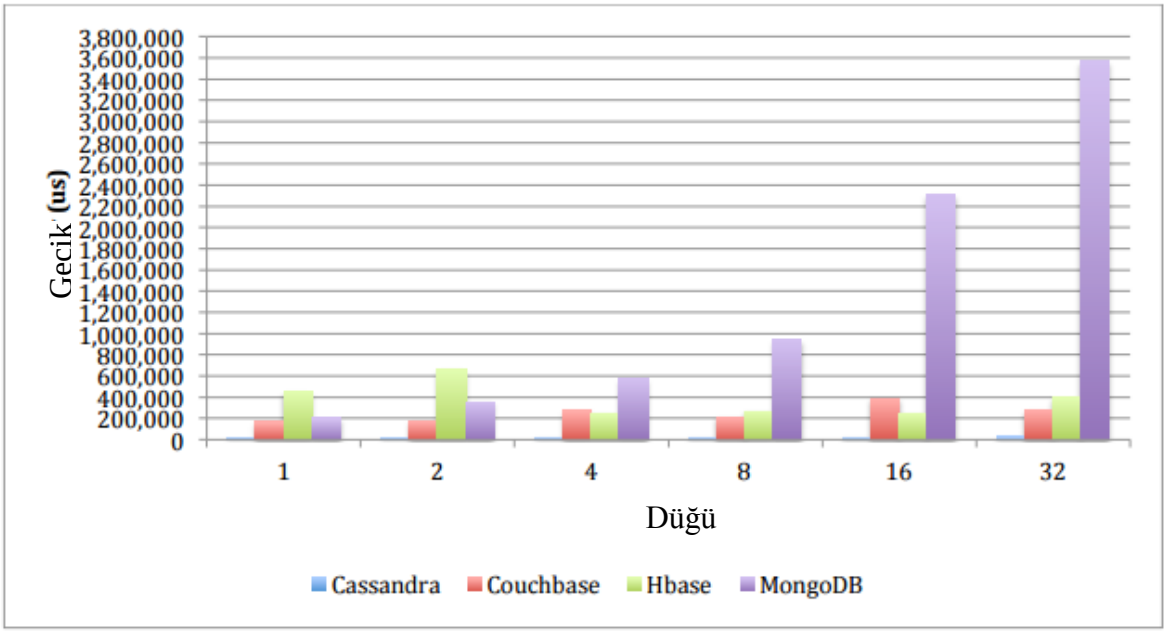
Projelere ait normalize edilmiş tanımlı veriler, toplu görüntüler üzerinde normalize edilmemiş formda saklanmalıdır. Bu şekilde sorgulara daha hızlı cevap verilmesi sağlanacaktır.

Hız Katmanı

Hız katmanı, toplu görüntülere henüz yansıtılamamış, sisteme yeni girilen verilerin saklandığı, ön işlemlerden geçirilerek sorgulamaya hazır hale getirildiği katmandır. Hız katmanı için kullanılacak araç katmanın sorumlu olduğu görevler gereğince artırımlı mimariye sahip olmalıdır. Yani seçilecek araç, gerçek zamanlı olarak anlık yazabilen ve yazılan veri üzerinde hızlı işlem yapabilen bir araç olmalıdır. Ana veri setindeki normalize edilmiş formdaki verileri, normalize edilmemiş forma getirebilmesi için, Map/Reduce algoritmasının seçilen araç tarafından desteklenmesi, gerçek zamanlı verilerin daha hızlı bir şekilde işlenmesini sağlar. Veriler arasında coğrafik veri tipleri de olduğu için seçilen araç, coğrafik veriler saklayabilmeli ve o veriler üzerinde işlem yapabilmelidir. Sayısal veriler sisteme atıldıktan sonra, tile katmanları oluşturulup harita üzerinde servis

edilmelidir. Kullanılacak araç GeoServer ile de entegre çalışabilmelidir. Bu yüzden önerilen araç CassandraDB'dir. CassandraDB bu özelliklerinin yanında oldukça hızlı çalışan bir araçtır. Çizelge 3.1'de diğer NoSQL veritabanları ile okuma ve yazma işlemlerinin karışık performans analizinin sonuçları görülmektedir.

Çizelge 3.1. NoSQL veritabanlarının performanslarının karşılaştırılması[26]



CassandraDB, Geomesa ile coğrafi verileri depolayabilmektedir. Geomesa; coğrafi verilerin depolanmasını ve depolanan veriler üzerinde işlem yapılabilmesini sağlayan, dağıtık mimariye sahip veri tabanları üzerine kurulan açık kaynak kodlu bir veritabanıdır. Postgis'in PostgreSQL'e sağladığı fonksiyonalitenin bir benzerini, Geomesa üzerine kurulduğu dağıtık mimariye sahip veritabanlarına sağlamaktadır. Ayrıca Geomesa'nın Geoserver eklentisi bulunmaktadır [27].

Sunum Katmanı

Sunum katmanı, toplu görüntülerin sorgulandığı katmandır. Sunum katmanı, Lambda mimarisinde toplu işlemlerin en son aşamasıdır. Bu katman verilerin servisinden sorumlu olduğu için, seçilecek araç toplu görüntüler arasında indeksleme yapabilmelidir.

Sunum katmanı için önerilen araç MongoDB'dir. MongoDB coğrafi verilerin ikincil indekslenmesine izin vermektedir. Bu durum çok karışık coğrafi sorgulara daha hızlı cevap verilmesini sağlamaktadır [28].

Kullanılacak aracı seçerken dikkat edilecek hususlardan bir diğeri de; veriler Geoserver ile sunulacağından, kullanılacak aracın Geoserver ile entegre olabilmesidir. Bu entegre ile, hem coğrafi veriler Geoserver tarafından hazırlanıp sunum katmanına aktarılabilecek hem de seçilen araç üzerinden coğrafi verilerin sunulması sağlanacaktır. MongoDB, resmi sitesinde Geoserver ile entegre olabilmesini sağlayan bir eklenti sunmaktadır.

3.2.4. Lambda mimarisinin dezavantajları

Lambda mimarisi farklı platformların birlikte çalıştığı karma bir veri çözümüdür. Sistemin doğru çalışabilmesi için mimaride kullanılacak platformlar birbiri ile senkronize çalışmalıdır. Bu senkronizeyi sağlayabilmek için de mimariyi hazırlayacak olan veritabanı uzmanlarının kullanacakları platformlarına detaylı bir şekilde hakim olması gerekmektedir.

Ayrıca iki farklı dağıtık mimaride çalışan veritabanından aynı sonucu üretmek için iki farklı kod yazılmalıdır. Toplu katmanda bulunan ana veri setinden servis katmanında kullanılacak toplu görüntüleri oluşturmak için yazılacak kod, hız katmanında güncel veriden gerçek zamanlı görüntüleri oluşturmak için de yazılmalıdır [12].

Seçilecek platformlar genellikle dağıtık yapıya sahip karmaşık araçlardır. Katmanlarda çalışacak farklı araçları senkronize edebilmek ve bu araçlardan aynı veriyi elde edebilmek için yazılacak kodun zorluğu sistemin dezavantajlarıdır.

SONUÇ VE ÖNERİLER

Günümüzde birçok kurum sosyal medya, bloglar, arşiv sistemleri gibi çok farklı kaynaklardan gelen, büyük miktardaki veriler ile uğraşmaktadır. Büyük miktarlardaki ve biçimlendirilmemiş tipleri de barındıran verileri yönetmek oldukça zor ve karmaşık bir iştir.

Coğrafi veriler, doğası itibariyle yoğun verilerdir. Bu yüzden gerçek zamanlı coğrafi veri sistemlerinde, milyonlarca kullanıcı olmadan da anlık olarak çok fazla sayıda yazma ve okuma işlemleriyle uğraşılmaktadır. İL-CAS projesinde yaşanan sorunlar anlık olarak gelen verinin büyüklüğü, gelen verilerin işlenerek gerçek zamanlı olarak servis edilmesi için gereken işlem fazlalığı ve sayısal veriler üzerinde ön belleğe alma işlemi yapılırken sistemin işlem yoğunluğundan dolayı cevap verememe ihtimalidir. İL-CAS'ta yaşanan sorunları çözmek için tek bir depolama araçı kullanılmasının çözüm sağlaması için çok yüksek maliyetli donanım yatırımları gerekmektedir. Bu yüzden İL-CAS'ın sorunlarının çözümünü sağlayacak farklı bir veri mimarisi araştırılmıştır.

Tezin ilk bölümünde geleneksel veritabanlarının ve artırımlı mimariye sahip platformların büyük miktarlardaki verileri yönetmekte yaşadığı problemler ile büyük veri sistemlerinde istenilen özellikler araştırılmıştır. Geleneksel veritabanları doğası itibariyle dağıtık yapıya sahip değillerdir. Bu yüzden bu veritabanlarını ölçeklendirmek oldukça zordur.

Geleneksel veritabanlarında büyük miktarlarda veri bulunduran tabloları ölçeklendirmek için, tablolar parçalara ayrılmaktadır. Ayrılan tabloların uygulama kodu ile senkronize edilmesi gerekmektedir. Bu işlemde yapılacak herhangi bir hata verilerin yanlış parçalara kaydedilmesine neden olabilmektedir. Geleneksel veritabanlarının ölçeklendirmede yaşadığı başarısızlık, hata toleransına sahip olmaması gibi nedenlerle büyük miktardaki verileri yönetmek için yetersiz kalmaktadır.

NoSQL veritabanları dağıtık mimariye sahip olduklarından yatay ölçeklendirmeye olanak sağlasalar da, tek bir aracın kullanılması İL-CAS'ta karşılaşan sorunları çözmekte yetersiz kalabileceği kanısına varılmıştır. Çünkü İL-CAS'ta yaşanan sorunun temelinde tek bir depolama aracına fazla işin yüklenmesinden kaynaklanan performans kaybı

yatmaktadır. Bu sorunu çözmek için iş yükünü parçalara ayırarak çözebilecek farklı bir veri mimarisi araştırılmıştır.

Tezin ikinci bölümünde, Lambda mimarisi incelenmiştir. Lambda mimarisi, tek bir depolama aracına bağlı kalmadan, birden fazla veritabanının belli amaçlar için kullanılmasına olanak veren bir veri sistem mimarisidir. Mimari, gerçek zamanlı sistemlerde, hem yeni hem de eski veriler üzerinde işlem yapabilmekte ve sorgulara düşük gecikme ile cevap verilebilmektedir.

Tezin son bölümünde; İL-CAS tanıtılmış, Lambda mimarisinin İL-CAS'a uygulanabilirliği irdelenmiştir. Ayrıca bu bölümde Lambda mimarisinde kullanılabilecek depolama araçları önerilmiştir. Kullanılacak araçların seçiminde, İL-CAS'ın teknik özelliklerine göre belirlenen kriter göz önünde bulundurulmuştur.

Tezde yapılan araştırmalar sonucunda Lambda mimarisinin kullanılmasının, İL-CAS'ta yaşanılan performans sorununa çözüm sağlayabileceği teorik olarak görülmüştür. İL-CAS ile yaşanan sorunların benzerlerinin Twitter ve Amazon gibi çok büyük miktarlardaki veriler ile uğraşan kurumların sorunlarına bu mimariyi kullanarak çözüm sağlaması, tezde önerilen sistemi destekler niteliktedir. Fakat Lambda mimarisi kullanımının, pratikte İL-CAS'a sağlayacağı performansı görebilmek için bu mimari pratiğe dökülmelidir.

KAYNAKLAR

1.Marz, N., and Warren J. (2015). *Big Data Principles and best practices of scalable realtime data systems* (Seventy Edititon). New York, NY: Manning, 1-225.

2.İnternet: Database Partitioning. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fstackoverflow.com%2Fquestions%2F20388923%2Fdatabase-partitioning-horizontal-vs-vertical-difference-between-normalizatio&date=2016-11-17, Son Erişim Tarihi: 11.17.2016.

3.İnternet: Advanteges and Disadvantages of DDBMS URL: http://www.webcitation.org/query?url=http%3A%2F%2Fecomputernotes.com%2Fdatabase-system%2Fadv-database%2Fadvantages-and-disadvantages-of-ddbms&date=2016-11-17, Son Erişim Tarihi: 11.17.2016.

4.İnternet: Relational Databases Are Not Designed For Scale URL: http://www.webcitation.org/query?url=http%3A%2F%2Fwww.marklogic.com%2Fblog%2Frelational-databases-scale%2F&date=2016-11-17,Son Erişim Tarihi: 17.11.2016.

5.Gilbert, S., Lynch, N.A. (2012) Perspectives on the CAP Theorem, MIT Institute of Electrical and Electronics Engineers.

6.Bhalla, A., Arora, R. (2014) Big Data Analysis and its Comparison with RDBMS, International Journal of Engineering Research & Technology Volume: 3 Issue:1.

7.İnternet: Key Value Store URL: http://www.webcitation.org/query?url=http%3A%2F%2Fwww.aerospike.com%2Fwhat-is-a-key-value-store%2F+&date=2016-11-17, Son Erişim Tarihi: 17.11.2016.

8.İnternet: Document Oriented Database URL: http://www.webcitation.org/query?url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FDocument-oriented_database&date=2016-11-17, Son Erişim Tarihi: 17.11.2016.

9.İnternet: Relational Databased URL: http://www.webcitation.org/query?url=http%3A%2F%2Fwww.agiledata.org%2Fessays%2FrelationalDatabases.html&date=2016-11-17, Son Erişim Tarihi: 17.11.2016.

10.İnternet: Big Data Tutorial MapReduce URL: http://www.webcitation.org/query?url=https%3A%2F%2Fwikis.nyu.edu%2Fdisplay%2FNYUHPC%2FBig%2BData%2BTutorial%2B1%253A%2BMapReduce&date=2016-11-17 , Son Erişim Tarihi: 17.11.2016.

11.İnternet: Hadoop MapReduce URL: http://www.webcitation.org/query?url=http%3A%2F%2Fdevveri.com%2Fhadoop%2Fmapreduce-nedir&date=2016-11-17, Son Erişim Tarihi: 17.11.2016.

12.İnternet: Questioning Lambda Architecture URL: http://www.webcitation.org/query?url=https%3A%2F%2Fwww.oreilly.com%2Fideas%2Fquestioning-the-lambda-architecture&date=2016-11-17, Son Erişim Tarihi: 17.11.2016.

13.İnternet: NoSQL Veritabanları URL: http://www.webcitation.org/query?url=http%3A%2F%2Fe-bergi.com%2Fy%2Fnosql-veritabanlari&date=2016-11-17, Son Erişim Tarihi: 17.11.2016.

14.İnternet: MongoDB Kullanan Girişimciler URL: http://www.webcitation.org/query?url=https%3A%2F%2Ftr.wikipedia.org%2Fwiki%2FMongoDB%23MongoDB_kullanan_giri.C5.9Fimciler&date=2016-11-17, Son Erişim Tarihi: 17.11.2016.

15.İnternet: Applying Lambda Architecture URL: http://www.webcitation.org/query?url=http%3A%2F%2Fwww.drdobbs.com%2Fdatabase%2Fapplying-the-big-data-lambda-architectur%2F240162604&date=2016-11-17, Son Erişim Tarihi: 17.11.2016.

16.İnternet: CAP Teoremi URL: http://www.webcitation.org/query?url=https%3A%2F%2Fmedium.com%2Fturkce%2Fcap-teoremi-nedir-49a23e6d2e10%23.klfdnmxke&date=2016-11-18, Son Erişim Tarihi: 17.11.2016.

17.İnternet: CAP Twelve Years Later: How the "Rules" Have Changed URL: http://www.webcitation.org/query?url=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fcap-twelve-years-later-how-the-rules-have-changed&date=2016-11-17, Son Erişim Tarihi: 17.11.2016.

18.İnternet: PostgreSQL VERİTABANI SUNUCUSU URL: http://www.webcitation.org/query?url=https%3A%2F%2Fwww.gunduz.org%2Fseminer%2Fpg%2FPostgreSQL-8.0_DevrimGunduz_29112004.pdf&date=2016-11-17, Son Erişim Tarihi: 17.11.2016.

19.İnternet: PostGIS URL: http://www.webcitation.org/query?url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FPostGIS&date=2016-11-17, Son Erişim Tarihi: 17.11.2016.

20.İnternet: Raster ve Vektör Veri Yapıları URL: http://www.webcitation.org/query?url=http%3A%2F%2Fportal.netcad.com.tr%2Fpages%2Fviewpage.action%3FpageId%3D106727005&date=2016-11-17, Son Erişim Tarihi: 17.11.2016.

21.İnternet: GeoServer About URL: http://www.webcitation.org/query?url=http%3A%2F%2Fgeoserver.org%2Fabout%2F&date=2016-11-18, Son Erişim Tarihi: 18.11.2016.

22.İnternet: Lambda Architecture URL: http://www.webcitation.org/query?url=http%3A%2F%2Flambda-architecture.net%2F&date=2016-11-18, Son Erişim Tarihi: 18.11.2016.

23.İnternet: Handling Five Billion Sessions In A Day URL: http://www.webcitation.org/query?url=https%3A%2F%2Fblog.twitter.com%2F2015%2Fhandling-five-billion-sessions-a-day-in-real-time&date=2016-11-18, Son Erişim Tarihi: 18.11.2016.

24.İnternet: How Smartnews Built A Lambda Architecture On Aws To Analyze Customer Behavior And Recommend Content URL: http://www.webcitation.org/query?url=https%3A%2F%2Faws.amazon.com%2Fblogs%2Fbig-data%2Fhow-smartnews-built-a-lambda-architecture-on-aws-to-analyze-customer-behavior-and-recommend-content%2F&date=2016-11-18, Son Erişim Tarihi: 18.11.2016.

25.İnternet: Massively Scalable NoSQL URL: http://www.webcitation.org/query?url=https%3A%2F%2Fcloud.google.com%2Fbigtable%2F&date=2016-11-18, Son Erişim Tarihi: 18.11.2016.

26.İnternet: Benchmarking Top NoSQL Databases URL: http://www.webcitation.org/query?url=http%3A%2F%2Fwww.datastax.com%2Fwp-content%2Fthemes%2Fdatastax-2014-08%2Ffiles%2FNoSQL_Benchmarks_EndPoint.pdf&date=2016-11-18, Son Erişim Tarihi: 18.11.2016.

27.İnternet: Get to the Point with Big Spatial Data URL: http://www.webcitation.org/query?url=http%3A%2F%2Fwww.ccri.com%2F2015%2F07%2F29%2Fget-to-the-point-with-big-spatial-data%2F&date=2016-11-18, Son Erişim Tarihi: 18.11.2016.

28.İnternet: MongoDB and HBase Compared URL: http://www.webcitation.org/query?url=https%3A%2F%2Fwww.mongodb.com%2Fcompare%2Fmongodb-hbase&date=2016-11-18, Son Erişim Tarihi: 18.11.2016.

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, Adı	:	BAMYACI, Hatice Tül Kübra
Uyruğu	:	T.C.
Doğum Tarihi ve Yeri	:	13.01.1990 Çorum
Medeni Hali	:	Evli
Telefon	:	0 (312) 303 36 24
Faks	:	-
e-mail	:	kcokan@hotmail.com

Eğitim

Derece	**Eğitim Birimi**	**Mezuniyet tarihi**
Lisans	Ankara Üniversitesi / Mühendislik Fakültesi / Bilgisayar Mühendisliği	2012
Lise	Bahçelievler Deneme Lisesi	2006

İş Deneyimi

Yıl	**Yer**	**Görev**
2013-Halen	İller Bankası Genel Müdürlüğü	Teknik Uzman Yrd.

Yabancı Dil

İngilizce

İLBANK

TÜRKİYE'NİN YAPICI GÜCÜ