Bölüm

# 14 Büyük Veri ve NoSQL

## Öğrenme Hedefleri

Bu bölümü tamamladıktan sonra şunları yapabileceksiniz:

**14-1** Büyük Verinin modern iş dünyasındaki rolünü açıklama

**14-2** Büyük Verinin temel özelliklerinin geleneksel "3 V"nin ötesine nasıl geçtiğini açıklama

**14-3** Hadoop frameworkünün temel bileşenlerinin nasıl çalıştığını açıklayabilme

**14-4** Hadoop ekosisteminin ana bileşenlerini tanımlama

**14-5** NoSQL veri modelinin dört ana yaklaşımını özetleme

**14-6** NoSQL veri modellerinin ilişkisel modelden nasıl farklı olduğunu tanımlama

**14-7** NewSQL veritabanlarının özelliklerini tanımlama

**14-8** MongoDB gibi bir belge veritabanının verileri nasıl depoladığını ve işlediğini açıklama

**14-9** Neo4j gibi bir grafik veritabanının verileri nasıl depoladığını ve işlediğini açıklayabilme

## ÖN İZLEME

Bölüm 2, Veri Modelleri, size gelişmekte olan NoSQL veri modelini ve NoSQL'in gelişmesine yol açan Büyük Veri sorununu tanıttı. Bu bölümde, bu konuları daha ayrıntılı olarak öğreneceksiniz. Büyük Verinin ve modern işletmeler için teşkil ettiği sorunun, Bölüm 2'de tanıtılan hacim, hız ve çeşitlilik ("3 V") özelliklerinden çok daha fazlası olduğunu göreceksiniz. Aslında, bu özelliklerin kendilerinin daha önce tartışılandan daha karmaşık olduğunu göreceksiniz.

Büyük Veri konularını öğrendikten sonra, Büyük Veriyi ele almak için geliştirilen ve geliştirilmeye devam eden teknolojiler hakkında bilgi edinirsiniz. İlk olarak, Hadoop frameworkündeki düşük seviyeli teknolojiler hakkında bilgi edinirsiniz. Hadoop, kuruluşların Büyük Veri'yi ele alma çabalarında standart bir bileşen haline geldi. Sonra ilişkisel olmayan veri modelinin geliştirilmesine yönelik NoSQL veri modelinin üst düzey yaklaşımları hakkında anahtar-değer veritabanları, belge veritabanları gibi veritabanları, sütun odaklı veritabanları ve grafik veritabanlarını öğreneceksiniz. Ayrıca ilişkisel veri tabanı sistemleri ile NoSQL arasındaki boşluğu dolduran NewSQL veritabanları hakkında da bilgi sahibi olursunuz.

Son olarak, mevcut iki NoSQL ürünündeki temel veritabanı etkinliklerini keşfedeceksiniz:

MongoDB ve Neo4j. İlişkisel veritabanlarında olduğu gibi, veri yönetimini gerçekleştirme yeteneği (yeni verileri depolama, mevcut verileri güncelleme, eski verileri kaldırma ve belirli verileri almak NoSQL veritabanlarının anahtarıdır. Çevrimiçi Ekler P ve Q sırasıyla MongoDB ve Neo4j için uygulamalı kodlama eğitimleri sağlar.

## Veri Dosyaları ve Mevcut Formatlar

| Dosya adı | Biçim/Açıklama |
|---|---|
| Ch14_FACT.json | MongoDB örneğinde kullanılan JavaScript Object Notation dosyası |
| Ch14_FCC.txt | Neo4j örneğinde kullanılan metin dosyası |

Veri Dosyaları cengage.com'da Mevcuttur

İlişkisel veritabanı modeli onlarca yıldır baskındır ve bu süre zarfında nesne yönelimli veritabanları ve veri ambarlarının geliştirilmesi gibi zorluklarla karşılaşmıştır. İlişkisel model ve buna dayalı araçlar bu zorluklara uyum sağlamak için evrim geçirmiş ve veri yönetimi arenasında baskın olmaya devam etmiştir. Her durumda, teknolojik ilerlemeler işletmelerin neyin mümkün olduğuna dair algılarını değiştirdiği ve kuruluşların artan veri kaldıracından değer yaratmaları için yeni fırsatlar yarattığı için zorluklar ortaya çıkmıştır. Bu zorlukların en sonuncusu Büyük Veri'dir. Büyük Veri, yeni bir veri depolama ve manipülasyon olanakları ve gereksinimleri dalgasını tanımlayan, tam olarak tanımlanmamış bir terimdir. Kuruluşların bu yeni veri dalgasını depolama, işleme ve analiz etme çabaları, veritabanı alanında ortaya çıkan en acil trendlerden birini temsil etmektedir. Büyük Veri dalgasıyla başa çıkmanın zorlukları, ilişkisel modelin altında yatan varsayımların çoğunu reddeden NoSQL veritabanlarının geliştirilmesine yol açmıştır. Büyük Veri terimi tutarlı bir tanımdan yoksun olsa da genellikle bir dizi özellik bu terimle ilişkilendirilmektedir.

# 14-1 Büyük Veri

Büyük Veri, hacim, hız ve çeşitlilik (3 V) özelliklerini, veriyi ilişkisel bir veritabanı yönetim sistemi tarafından yönetilmeye uygun hale getirmeyecek ölçüde sergileyen bir veri kümesini ifade eder. Bu özellikler aşağıdaki gibi tanımlanabilir:

- Hacim - depolanacak veri miktarı
- Hız - verilerin sisteme giriş hızı
- Çeşitlilik - depolanacak verilerin yapısındaki varyasyonlar

**hacim**
Büyük Veri'nin depolanacak veri miktarını tanımlayan bir özelliği.

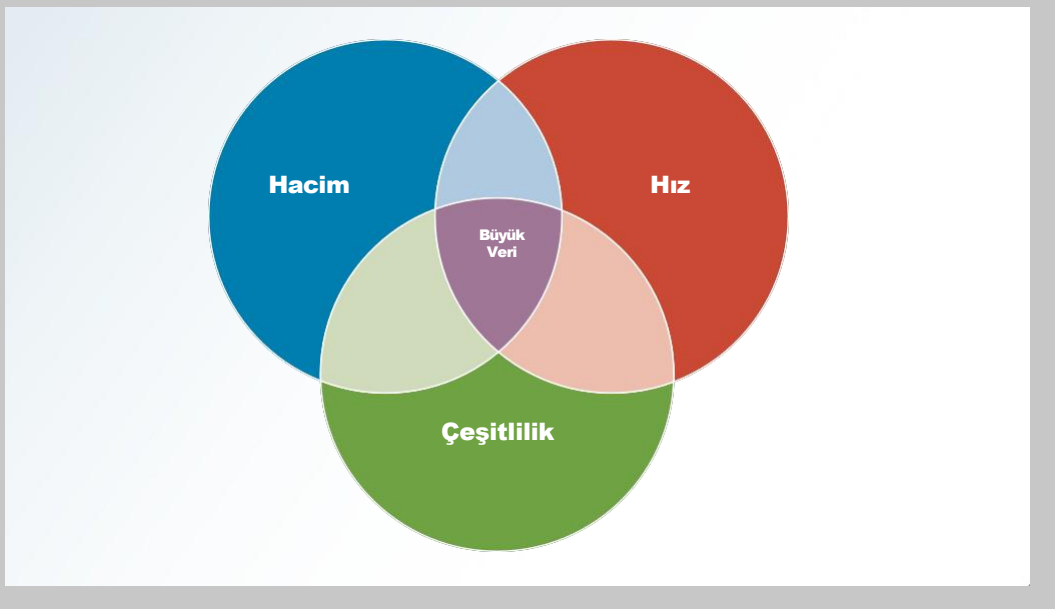
**hız**
Verilerin sisteme girme ve işlenme hızını tanımlayan bir Büyük Veri özelliği.

**çeşitlilik**
Büyük Veri'nin, depolanacak verilerin yapısındaki varyasyonları tanımlayan bir özelliği.

Bu özelliklerle ilişkili belirli değerlerin eksikliğine dikkat edin. Büyük Veri'nin tanımlanmasında belirsizliğe yol açan da bu spesifiklik eksikliğidir. Beş yıl önce Büyük Veri olan bir şey şimdi Büyük Veri olarak kabul edilmeyebilir. Benzer şekilde, şu anda Büyük Veri olarak kabul edilen bir şey, bundan beş yıl sonra Büyük Veri olarak kabul edilmeyebilir. Önemli olan, mevcut ilişkisel veritabanı teknolojisinin verileri yönetmekte zorlanacağı ölçüde özelliklerin mevcut olmasıdır.

Büyük Verinin tanımlanması sorununa bir yenisi daha eklenerek, bir veri setinin Büyük Veri olarak kabul edilebilmesi için 3 V'den hangisinin bulunması gerektiği konusunda uzmanlar arasında görüş ayrılıkları ortaya çıkmıştır. Başlangıçta Büyük Veri, Şekil 14.1'de gösterildiği gibi 3 V'nin bir kombinasyonu olarak düşünülmüştür. Karmaşık yapılarda bir araya getirilen metin, grafik, video ve ses kaynaklarının bir kombinasyonu olan Web verileri, bu üç özelliği de içeren veri yönetimi için yeni zorluklar yaratmıştır.

**Şekil 14.1** Büyük Verinin Orijinal Görünümü

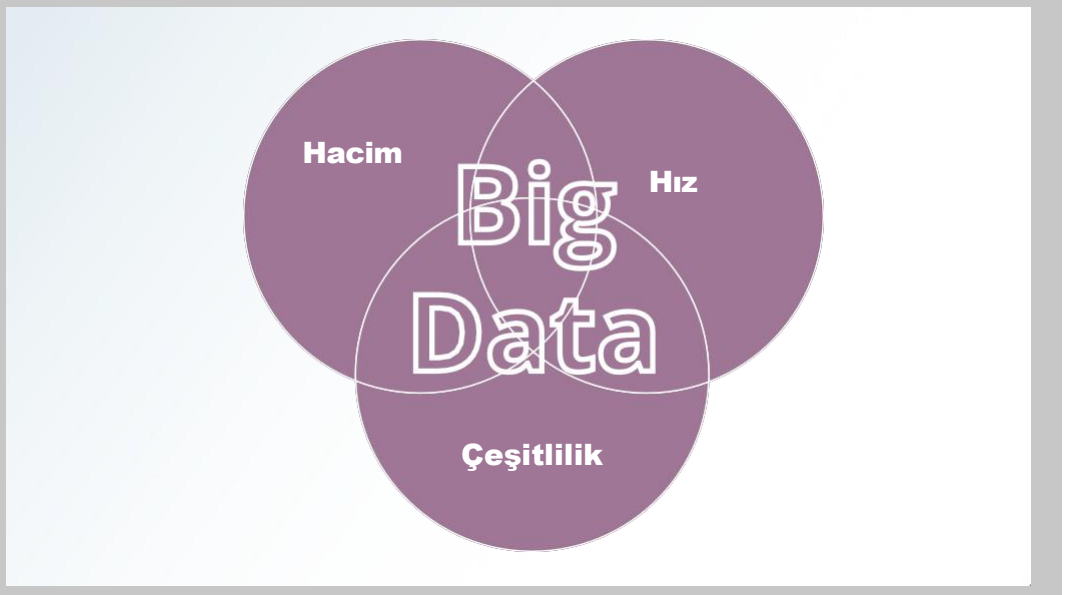



1990'larda dot-com (.com) balonunun patlamasından sonra, birçok yeni kurulmuş web tabanlı şirket başarısız oldu ancak hayatta kalan şirketler, web ticaretinin daha küçük bir işletme grubunda birleşmesiyle önemli bir büyüme yaşadı. Sonuç olarak, Google ve Amazon gibi şirketler önemli bir büyüme yaşadı ve Büyük Veri yönetiminin baskısını ilk hissedenler arasında yer aldı. Sosyal medya devi Facebook'un başarısı da bunu takip etti ve bu şirketler Büyük Veri sorunlarına yönelik teknolojilerin yaratılmasında öncü oldular. Google BigTable veri deposunu, Amazon Dynamo'yu ve Facebook bu bölümün ilerleyen kısımlarında tartışılacak olan Cassandra teknolojilerini, 3 V'nin özelliklerine sahip büyük miktarda veriyi depolama ve yönetme ihtiyacının artması bu sorunlarla başa çıkması için yarattı.

Buna rağmen sosyal medya ve web verileri Büyük Veri sorunlarının algılanmasında ön planda olsa da diğer kuruluşların da Büyük Veri sorunları vardır. Daha yakın zamanlarda, teknolojideki değişiklikler işletmelerin veri üretme ve izleme fırsatlarını artırmış, böylece Büyük Veri, Şekil 14.2'de gösterildiği gibi, 3 V'nin hepsini değil ama herhangi birini içerecek şekilde yeniden tanımlanmıştır. Teknolojideki ilerlemeler, belirli alanlarda büyümeyi teşvik edebilecek çok sayıda kullanıcı tarafından oluşturulan verinin ve makine tarafından oluşturulan verinin ortaya çıkmasına neden olmuştur.

Örneğin Disney World, park ziyaretçilerinin bileklerine takmaları için "Sihirli Bantlar "ı tanıttı. Her ziyaretçinin Sihirli Bandı, Disney'in o kişi hakkında depoladığı verilerin çoğuna bağlıdır. Bu bantlar radyo frekansı tanımlama (RFID) ve yakın alan iletişimi (NFC) kullanarak park içindeki gezintiler için bilet, otel odası anahtarı ve hatta kredi kartı görevi görüyor. Bantlar takip edilebilir, böylece Disney sistemleri insanları parkta ilerlerken izleyebilir ve hangi Disney karakterleriyle (onlar da izlenir) etkileşime girdiklerini, yapılan alışverişleri, kuyruklarda bekleme sürelerini ve daha fazlasını kaydedebilir. Ziyaretçiler akıllı telefonlarındaki Disney uygulaması aracılığıyla restoranlarda rezervasyon yaptırabilir ve yemek siparişi verebilirler ve Sihirli Bantları takip ederek restoran personeli ziyaretçilerin rezervasyon için ne zaman geldiklerini bilir, hangi masaya oturduklarını takip edebilir ve konukların oturmasından birkaç dakika sonra yemeklerini teslim edebilir. Disney ayrıca parkın dört bir yanına yerleştirdiği çok sayıda kamerayla ziyaretçilerin parkta kaldıkları süre boyunca fotoğraflarını ve kısa videolarını çekerek tatil deneyimlerinin kişiselleştirilmiş bir filmini oluşturabiliyor ve bu filmler daha sonra ziyaretçilere hediyelik eşya olarak satılabiliyor. Tüm bunlar, her bir banttan gerçek zamanlı olarak işlenen sürekli bir veri akışının yakalanmasını gerektirir. Disney World'de her gün her biri bir Magic Band'e sahip on binlerce ziyaretçi olduğu düşünüldüğünde, verilerin hacmi, hızı ve çeşitliliği muazzamdır.

**Şekil 14.2** Büyük Veriye Güncel Bakış




## 14-1a Hacim

Depolanacak veri miktarı olan hacim, Büyük Veri'nin temel bir özelliğidir. Büyük Veri ile ilişkili depolama kapasiteleri son derece büyüktür. Tablo 14.1'de veri depolama kapasitesi birimleri için tanımlar verilmektedir.

**Tablo 14.1** Depolama Kapasite Birimleri

| Dönem | Kapasite | Kısaltma |
|---|---|---|
| Bit | 0 veya 1 değeri | b |
| Bayt | 8 bit | B |
| Kilobayt | 1024* bayt | KB |
| Megabayt | 1024 KB | MB |
| Gigabayt | 1024 MB | GB |
| Terabayt | 1024 GB | TB |
| Petabayt | 1024 TB | PB |
| Eksabayt | 1024 PB | EB |
| Zettabayt | 1024 EB | ZB |
| Yottabayt | 1024 ZB | YB |

*Bitlerin ikili yapıda olması ve diğer tüm depolama değerlerinin dayandığı temel olması nedeniyle, veri depolama birimlerine ilişkin tüm değerlerin 2'nin kuvvetleri cinsinden tanımlandığını unutmayın. Örneğin, *kilo* öneki tipik olarak 1000 anlamına gelir; ancak veri bir kilobayt = $2^{10}$ = 1024 bayttır.

Doğal olarak, depolanması gereken veri miktarı arttıkça, daha büyük depolama cihazlarına olan ihtiyaç artar. Bu durumda sistemler, ölçeği ya büyütebilir ya da ölçek küçültebilir. **Ölçek büyütme**, aynı sayıda sistemi korumak ancak her bir sistemi daha büyük bir sisteme geçirmektir: örnek olarak 16 CPU çekirdeği ve 1 TB depolama sistemi olan bir sunucudan 64 CPU çekirdeği ve 100 TB depolama sistemi olan bir sunucuya geçmek gibi. Ölçek büyütme, daha büyük ve daha hızlı sistemlere geçmeyi içerir. Ancak, tek bir sistemin ne kadar büyük ve hızlı olabileceğinin sınırları vardır. Ayrıca, bu yüksek güçlü sistemlerin maliyetleri dramatik bir oranda artmaktadır.

**ölçek büyütme**
Veri büyümesiyle başa çıkmak aynı yapının daha güçlü sistemlere taşınmasını içeren bir yöntem.

**ölçeklendirme**
Veri depolama yapılarının bir emtia sunucuları kümesine dağıtılmasını içeren veri büyümesiyle başa çıkma yöntemi.

Öte yandan, **ölçeklendirme**, iş yükü bir sunucunun kapasitesini aştığında, iş yükünün bir dizi sunucuya yayılması anlamına gelir. Bu aynı zamanda *kümeleme* olarak da adlandırılır - bir iş yükünü paylaşmak için düşük maliyetli sunuculardan oluşan bir küme *oluşturmak.* Tek bir 1 PB depolama sistemi satın almaktansa on adet 100 TB depolama sistemi satın almak daha ucuz olduğundan bu, bilgi işlem kaynaklarının toplam maliyetini azaltmaya yardımcı olabilir. Hiç kuşkunuz olmasın, kuruluşlar bu uç boyutlarda depolama kapasitelerine ihtiyaç duymaktadır. eBay gibi kuruluşlar, düzinelerce petabayta kolayca ulaşan tıklama akışı verileri toplamaktadır. Buna ek olarak kurumsal veri ambarları da petabayt boyutunda olabilir ve yüz binlerce düğüme yayılabilir.

Bölüm 3'ten, ilişkisel modelin temsil ettiği en büyük ilerlemelerden birinin, temel veri depolama ve manipülasyonunun karmaşıklığını kullanıcıdan gizleyebilen ve böylece verilerin her zaman tablolarda görünmesini sağlayan sofistike bir veritabanı yönetim sistemi olan RDBMS'nin (*ilişkisel veritabanı yönetim sistemi*) geliştirilmesi olduğunu hatırlayın. Bu fonksiyonları yerine getirmek için DBMS (*veritabanı yönetim sistemi*), veritabanı sisteminin "beyni" olarak hareket eder ve veritabanı içindeki tüm veriler üzerinde kontrolü elinde tutmalıdır. Bölüm 12'de tartışıldığı gibi, ilişkisel bir veritabanını çoğaltma ve parçalama kullanarak birden fazla sunucuya dağıtmak mümkündür. Ancak, DBMS'nin veritabanındaki tüm veriler için tek bir kontrol noktası olarak hareket etmesi gerektiğinden, veritabanını birden fazla sisteme dağıtmak sistemler arasında yüksek derecede iletişim ve koordinasyon gerektirir. Düğüm sayısı arttıkça iletişim ve koordinasyonun artan performans maliyetleri nedeniyle DBMS'yi dağıtma yeteneği ile ilgili önemli kısıtlamalar vardır. Bu, veri hacmi büyüdükçe ilişkisel bir veritabanının ölçeklendirilebilme derecesini sınırlar ve RDBMS'leri kümeler için uygunsuz hale getirir.

## Not

SQL Server ve Oracle Real-Application Clusters gibi bazı RDBMS ürünleri meşru bir şekilde kümeleri desteklediklerini iddia etseler de bu kümelerin kapsamı sınırlıdır ve genellikle depolama alanı ağı gibi tek bir paylaşılan veri depolama alt sistemine dayanırlar.

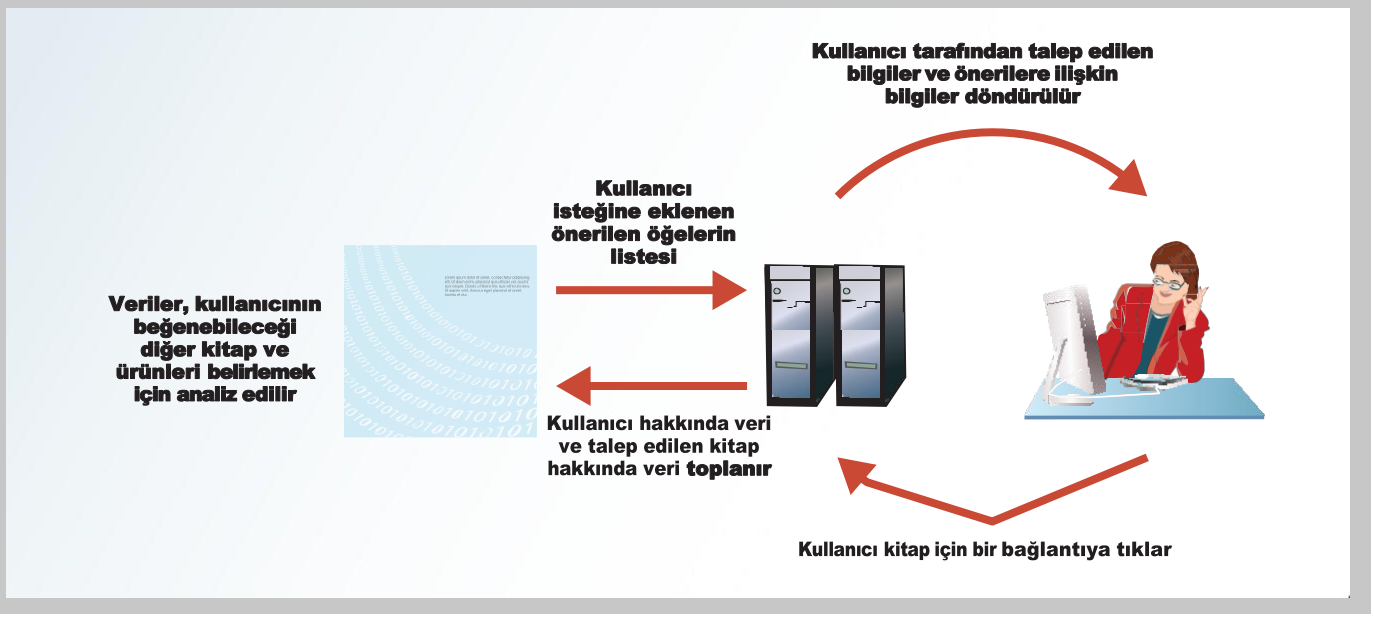
## 14-1b Hız

Büyük Veri'nin bir diğer temel özelliği olan hız, yeni verilerin sisteme girme hızının yanı sıra verilerin işlenmesi gereken hızı da ifade eder. Hızla ilgili sorunlar birçok yönden hacimle ilgili sorunların aynısıdır. Örneğin, Amazon gibi bir web perakendecisini düşünün. Geçmişte, bir perakende mağazası yalnızca satın alma işlemi yapan müşterinin son işlemiyle ilgili verileri yakalayabilirdi. Günümüzde Amazon gibi bir perakendeci yalnızca son işlemi değil, aynı zamanda arama, göz atma, karşılaştırma ve satın alma sürecindeki her fare tıklamasını da yakalamaktadır. Amazon, 20 dakikalık bir alışveriş deneyiminde tek bir olayı (nihai satış) yakalamak yerine, bu 20 dakikalık süre içinde 30 olayla ilgili verileri yakalayabilir; bu da veri hızında 30 kat artış anlamına gelir. RFID, GPS ve NFC gibi teknolojideki diğer gelişmeler, genellikle gerçek zamanlı olarak depolanması gereken büyük miktarda veri üreten yeni veri toplama fırsatları katmanları ekler. Örneğin, RFID etiketleri envanter ve depo yönetimi için öğeleri izlemek için kullanılabilir. Etiketler, etiket ile okuyucu arasında görüş hattı gerektirmez ve okuyucu, ürünler henüz kutulardayken yüzlerce etiketi aynı anda okuyabilir. Bu, üretilen bir ürünün belirli bir miktarını izlemek için tek bir kayıt yerine, her ürünün izlendiği anlamına gelir ve herhangi bir zamanda sisteme iletilen veri miktarında birkaç büyüklük sırası artışı yaratır.

Verilerin sisteme girme hızına ek olarak, Büyük Verinin eyleme dönüştürülebilir olması için bu verilerin çok hızlı bir şekilde işlenmesi gerekir. İşleme hızı iki kategoriye ayrılabilir.

- Akış işleme
- Geri bildirim döngüsü işleme

**Şekil 14.3** Geri Besleme Döngüsü İşleme




**Akış** işleme, girdi işlemeye odaklanır ve veri akışının sisteme girerken analiz edilmesini gerektirir. Bazı durumlarda, büyük hacimli veriler sisteme o kadar hızlı bir şekilde girebilir ki, tüm verileri depolamaya çalışmak mümkün değildir. Hangi verilerin saklanacağını ve hangi verilerin atılacağını belirlemek için veriler sisteme girerken işlenmeli ve filtrelenmelidir. Örneğin, dünyanın en büyük ve en güçlü parçacık hızlandırıcısı olan CERN Büyük Hadron Çarpıştırıcısı'nda deneyler saniyede yaklaşık 600 TB ham veri üretmektedir. Bilim insanları hangi verilerin saklanacağına önceden karar vermek için **algoritmalar** oluşturdular. Bu algoritmalar iki aşamalı bir süreçle uygulanarak verilerin saniyede sadece 1 GB'a kadar filtrelenmesi sağlanıyor ve bu veriler gerçekten saklanıyor.[1]

**Geri bildirim döngüsü işleme**, eyleme geçirilebilir sonuçlar üretmek için verilerin analizini ifade eder. Akış işleme girdilere odaklanmış olarak düşünülebilirken, geri bildirim döngüsü işleme çıktılara odaklanmış olarak düşünülebilir. Verilerin yakalanması, işlenerek kullanılabilir bilgi haline getirilmesi ve ardından bu bilgilere göre hareket edilmesi bir geri bildirim döngüsüdür. Şekil 14.3'te kitap satın alımları için öneriler sunan bir geri bildirim döngüsü gösterilmektedir. Anında sonuç sağlamak için geri bildirim döngüsü işleme, büyük miktarda verinin sadece birkaç saniye içinde analiz edilmesini gerektirir, böylece analiz sonuçları gerçek zamanlı olarak kullanıcıya sunulan ürünün bir parçası haline gelebilir. Tüm geri bildirim döngüleri sonuçları anında veri ürünlerine dahil etmek için kullanılmaz. Geri bildirim döngüsü işleme aynı zamanda kuruluşların terabaytlarca ve petabaytlarca veriyi eleyerek karar vericileri daha hızlı stratejik ve taktiksel kararlar almaları konusunda bilgilendirmelerine yardımcı olmak için de kullanılır ve veri analitiğinin önemli bir bileşenidir.

**akış işleme**
Depolamadan önce hangi verilerin saklanacağı ve hangi verilerin atılacağı konusunda karar vermek için veri girdilerinin işlenmesi.

**algoritma**
Bir hesaplamadaki işlem veya işlemler kümesi.

**geri bildirim döngüsü işleme**
Eyleme geçirilebilir sonuçlar üretmek için depolanan verileri analiz etme.

## 14-1c Çeşitlilik

Büyük Veri bağlamında çeşitlilik, verilerin yakalanabileceği çok çeşitli format ve yapıları ifade eder. Veriler yapılandırılmış, yapılandırılmamış veya yarı yapılandırılmış olarak değerlendirilebilir. **Yapılandırılmış veriler**, önceden tanımlanmış bir veri modeline uyacak şekilde düzenlenmiş verilerdir. **Yapılandırılmamış veri**, önceden tanımlanmış bir veri modeline uyacak şekilde düzenlenmemiş veridir. Yarı yapılandırılmış veriler her ikisinin de unsurlarını bir araya getirir; verilerin bazı kısımları önceden tanımlanmış bir modele uyarken diğer kısımları uymaz.

**yapılandırılmış veri**
Önceden tanımlanmış bir veri modeline uyan veriler.

**yapılandırılmamış veri**
Önceden tanımlanmış bir veri modeline uymayan veriler.

[1]CERN, "İşleme: Ne kaydedilmeli?" https://home.web.cern.ch/about/computing/processing-what-record, 20 Ağustos 2015.

İlişkisel veritabanları yapılandırılmış verilere dayanır. Bölüm 4'te tartışıldığı gibi, veritabanı tasarımcısı tarafından iş kurallarına dayalı olarak bir veri modeli oluşturulur. Veriler veritabanına girerken, veri modelinde tanımlandığı gibi ilgili tablolarda ve sütunlarda depolanmak üzere ayrıştırılır ve yönlendirilir. Kuruluşların kullandığı işlemsel verilerin çoğu yapılandırılmış bir ortamda iyi çalışsa da, dünyadaki verilerin çoğu yarı yapılandırılmış veya yapılandırılmamış verilerdir. Yapılandırılmamış veriler arasında haritalar, uydu görüntüleri, e-postalar, metinler, tweetler, videolar, transkriptler ve diğer birçok veri formu yer alır. İlişkisel modelin baskın olduğu on yıllar boyunca, ilişkisel veritabanları bazı yapılandırılmamış veri biçimlerini ele alacak şekilde gelişmiştir. Örneğin, büyük ölçekli RDBMS'lerin çoğu ses, video ve grafik verileri gibi yapılandırılmamış nesnelerin tek bir atomik değer olarak depolanmasına olanak tanıyan ikili büyük nesne (BLOB) veri türünü destekler. BLOB verileriyle ilgili bir sorun, verilerin anlamsal değerinin, yani nesnenin taşıdığı anlamın veri işleme tarafından erişilemez ve yorumlanamaz olmasıdır.

Büyük Veri, verilere bir veri modeli veya yapısı empoze etmeye çalışmadan, verilerin doğal olarak hangi formatta mevcutsa o formatta yakalanmasını gerektirir. Bu, ilişkisel bir veritabanında veri işleme ile Büyük Veri işleme arasındaki temel farklardan biridir. İlişkisel veritabanları, veriler yakalanıp depolandığında verilere bir yapı empoze eder. Büyük Veri işleme, geri alma ve işlemenin bir parçası olarak uygulamalar için ihtiyaç duyulduğunda verilere bir yapı dayatır. Erişim ve işleme sırasında yapı sağlamanın bir avantajı, verileri farklı uygulamalar için farklı şekillerde yapılandırabilme esnekliğidir.

## 14-1d Diğer Özellikler

Büyük Veri'yi 3 V ile tanımlamak oldukça standarttır. Ancak endüstri olgunlaştıkça, diğer özelliklerin de eşit derecede önemli olduğu öne sürülmüştür. Bu özellikler, 3 V'nin ruhuna uygun olarak, Tablo 14.2'de özetlendiği gibi, tipik olarak ek V olarak sunulmaktadır. **Değişkenlik**, bağlama bağlı olarak verilerin anlamındaki değişiklikleri ifade eder. *Çeşitlilik* ve *değişkenlik* benzer terimler olsa da Büyük Veri'de farklı anlamlara gelmektedir. Çeşitlilik yapıdaki farklılıklarla ilgilidir. Değişkenlik ise anlamdaki farklılıklarla ilgilidir. Değişkenlik özellikle kelimelerin anlamlarını anlamaya çalışan duygu analizi gibi alanlarda önemlidir. **Duygu analizi**, bir ifadenin bir konu hakkında olumlu, olumsuz veya nötr bir tutum sergileyip sergilemediğini belirlemeye çalışan bir metin analizi yöntemidir. Örneğin, şu ifadeler: "Yeni bir akıllı telefon aldım - bayıldım!" ve "Yeni akıllı telefonumun ekranı ilk düşürdüğümde paramparça oldu - bayıldım!" İlk ifadede "bayıldım" ifadesinin varlığı, bir algoritmanın ifadeyi olumlu bir tutum ifade ettiği şeklinde doğru bir şekilde yorumlamasına yardımcı olabilir. Ancak ikinci ifade olumsuz bir tutumu ifade etmek için alaycı bir dil kullanmaktadır, nedenle "bayıldım" ifadesinin varlığı analizin ifadenin anlamını yanlış yorumlamasına neden olabilir.

**değişkenlik**
Büyük Verinin özelliği, aynı veri değerlerinin zaman içinde anlamlarının değişmesidir.

**duygu analizi**
Bir ifadenin aşağıdakileri iletip iletmediğini belirlemeye çalışan bir metin analizi yöntemi olumlu, olumsuz veya nötr bir tutum.

**Tablo 14.2 Büyük Verinin Ek Avantajları**

| Karakteristik | Açıklama |
|---|---|
| Değişkenlik | Verilerin anlamı bağlama göre değişir. |
| Doğruluk | Veriler doğru. |
| Değer (Yaşayabilirlik) | Veriler anlamlı bilgiler sağlayabilir. |
| Görselleştirme | Veriler, anlaşılabilir hale getirilecek şekilde sunulabilir. |

**Doğruluk**, verilerin güvenilirliğini ifade eder. Karar alıcılar verilerin ve bunlardan üretilen bilgilerin doğruluğuna makul ölçüde güvenebilir mi? Veri yakalama ve bazı analizlerin otomasyonu göz önüne alındığında bu özellikle önemlidir. Verilerle ilgili belirsizlik, yüksek hız nedeniyle verilerin yalnızca seçilen kısımlarını yakalamak zorunda kalmak gibi çeşitli nedenlerden kaynaklanabilir. Ayrıca, duyarlılık analizi açısından, müşterilerin görüşleri ve tercihleri zaman içinde değişebilir, bu nedenle zamanın bir noktasındaki yorumlar zamanın başka bir noktasında eylem için uygun olmayabilir.

**doğruluk**
Bir veri kümesinin güvenilirliği.

**Değer**, Büyük Veri için giderek daha önemli bir özellik olarak lanse edilmektedir. *Uygulanabilirlik* olarak da adlandırılan değer, verilerin kuruma değer katabilecek anlamlı bilgiler sağlamak üzere analiz edilebilme derecesini ifade eder. Bir veri kümesinin yakalanabilir *olması*, yakalanması *gerektiği* anlamına gelmez. Yalnızca kurumsal davranışı etkileme potansiyeline sahip analizler için temel oluşturabilecek veriler bir şirketin Büyük Veri çabalarına dahil edilmelidir.

**değer**
Verilerin anlamlı içgörüler sağlamak üzere analiz edilebilme derecesi.

Büyük Veri'nin son özelliği görselleştirmedir. **Görselleştirme,** verileri anlaşılabilir kılacak şekilde grafiksel olarak sunma becerisidir. Veri yığınları karar vericileri gerçekler içinde boğabilir ancak bu gerçeklerin ne anlama geldiğini çok az anlayabilirler. Görselleştirme, gerçekleri sunmanın bir yoludur, böylece karar vericiler içgörü kazanmak için bilginin anlamını kavrayabilirler.

**görselleştirme**
Verileri kullanıcılar için anlaşılabilir kılacak şekilde grafiksel olarak sunma becerisi.

Bu ek V'nin Büyük Veri'nin özellikleri olmadığı ya da belki daha doğru bir ifadeyle *sadece* Büyük Veri'nin özellikleri olmadığı iddia edilebilir. Verilerin doğruluğu en küçük veri deposunda bile bir sorundur, bu nedenle veri yönetimi ilişkisel veritabanlarında çok önemlidir. Verilerin değeri, ilişkisel bir veritabanındaki geleneksel, yapılandırılmış veriler için de geçerlidir. Veri modellemenin anahtarlarından biri, yalnızca kullanıcıların ilgisini çeken verilerin veri modeline dahil edilmesi gerektiğidir. Değeri olmayan veriler, Büyük Veri olsun ya da olmasın hiçbir veri deposuna kaydedilmemelidir. Görselleştirme, genellikle RDBMS ürünlerinde yapılandırılmış veri depoları olarak tutulan veri ambarları ile çalışmada önemli bir araç olarak Bölüm 13'te uzun uzun tartışılmış ve gösterilmiştir. Unutulmaması gereken önemli nokta, ilişkisel modeldeki verilerle çalışırken önemli bir rol oynayan bu özelliklerin evrensel olduğu ve Büyük Veri için de geçerli olduğudur.

Büyük Veri, veri yönetimi zorluklarında yeni bir dalgayı temsil eder, ancak bu ilişkisel veritabanı teknolojisinin ortadan kalkacağı anlamına gelmez. Bölüm 10'da tartışıldığı gibi ACID (atomiklik, tutarlılık, izolasyon ve dayanıklılık) işlemlerine bağlı olan yapılandırılmış veriler, iş operasyonları için her zaman kritik olacaktır. İlişkisel veritabanları bu tür verilerin depolanması ve yönetilmesi için hala en iyi yoldur. Değişen şey, on yıllardır ilk kez ilişkisel veritabanlarının bir kuruluşun tüm verilerini depolamak ve yönetmek için en iyi yol olmayabileceğidir. İlişkisel modelin yükselişinden bu yana, veri yöneticilerinin yeni depolama gereksinimleriyle karşılaştıklarında verdikleri karar ilişkisel bir veritabanı kullanıp kullanmayacakları değil, hangi ilişkisel DBMS'yi kullanacaklarıydı. Şimdi, ilişkisel bir veritabanı kullanıp kullanmama kararı gerçek bir sorudur. Bu durum, bir kuruluşun altyapısında çeşitli veri depolama ve yönetim teknolojilerinin bir arada bulunması anlamına gelen **çok dilli kalıcılığa** yol açmıştır. Tartışıldığı gibi ölçek büyütme, ilişkisel veritabanları büyüdükçe genellikle uygulanabilir bir seçenek olarak kabul edilir. Ancak bunun pratik sınırları ve maliyet hususları vardır ve bu da onu birçok Büyük Veri kurulumu için uygulanamaz hale getirir. Düşük maliyetli emtia sunucularına dayalı kümeler halinde ölçeklendirme, kuruluşların şu anda Büyük Veri yönetimi için izlediği baskın yaklaşımdır. Sonuç olarak, ilişkisel modele dayanmayan yeni teknolojiler geliştirilmiştir.

**çok dilli kalıcılık**
Çeşitliliğin; veri depolama ve veri yönetim teknolojilerinin, bir kuruluşun bünyesindeki altyapının bir arada bulunması.

# 14-2 Hadoop

Büyük Veri, büyük ölçekli kümeler için tasarlanmış dağıtılmış veri depolamaya farklı bir yaklaşım gerektirir. Diğer uygulama teknolojileri mümkün olsa da Hadoop çoğu Büyük Veri depolama ve işleme için fiili standart haline gelmiştir. Hadoop bir veritabanı değildir. Hadoop, çok büyük veri kümelerini bilgisayar kümelerine dağıtmak ve işlemek için Java tabanlı bir çerçevedir. Hadoop çerçevesi birçok parça içermekle birlikte, en önemli iki bileşen Hadoop Dağıtılmış Dosya Sistemi (HDFS) ve MapReduce'dur. HDFS düşük seviyeli bir dağıtık dosya işleme sistemidir, yani doğrudan veri depolama için kullanılabilir. MapReduce, büyük veri kümelerinin son derece paralel, dağıtılmış bir şekilde işlenmesini destekleyen bir programlama modelidir. HDFS ve MapReduce'u ayrı ayrı kullanmak mümkün olsa da iki teknoloji birbirini tamamlar ve böylece bir Hadoop sistemi olarak birlikte daha iyi çalışırlar.

Hadoop, muazzam miktardaki verileri geniş sunucu kümelerine dağıtmak ve işlemek için özel olarak tasarlanmıştır.

## 14-2a HDFS

**Hadoop Dağıtılmış Dosya Sistemi (HDFS)** Büyük miktarda veriyi yüksek hızlarda yönetmek için tasarlanmış, yüksek oranda dağıtılmış, hataya dayanıklı bir dosya depolama sistemi.

Veri dağıtımına yönelik **Hadoop Dağıtılmış Dosya Sistemi (HDFS)** yaklaşımı birkaç temel varsayıma dayanmaktadır:

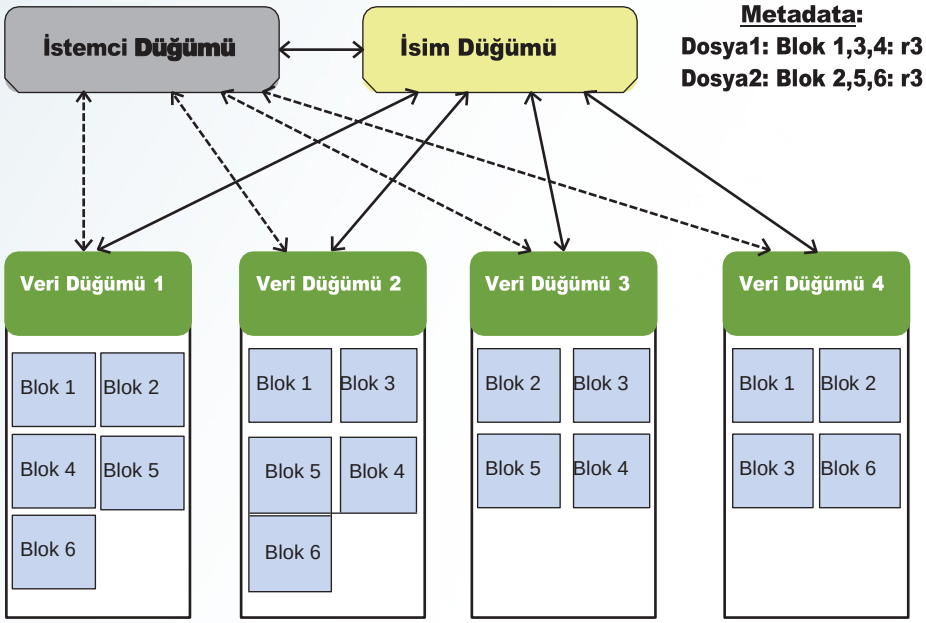
- *Yüksek hacim*. Büyük Veri uygulamalarındaki veri hacminin terabayt, petabayt veya daha büyük olması beklenir. Hadoop, HDFS'deki dosyaların son derece büyük olacağını varsayar. HDFS'deki veriler, tıpkı diğer dosya depolama türlerinde olduğu gibi fiziksel bloklar halinde düzenlenir. Örneğin, tipik bir kişisel bilgisayarda dosya depolama, ilgili donanım ve işletim sistemine bağlı olarak genellikle 512 bayt boyutunda bloklar halinde düzenlenir. İlişkisel veritabanları genellikle bunları veritabanı blokları halinde toplar. Oracle varsayılan olarak verileri 8 KB'lık fiziksel bloklar halinde düzenler. Öte yandan Hadoop'un varsayılan blok boyutu 64 MB'dir (bir Oracle bloğunun 8.000 katı!) ve daha da büyük değerlere göre yapılandırılabilir. Sonuç olarak, dosya başına blok sayısı büyük ölçüde azalır ve her dosyadaki blokları izlemek için meta veri ek yükü basitleştirilir.
- *Bir kere yaz, çok kere oku*. Bir kez yaz, çok oku modelinin kullanılması eşzamanlılık sorunlarını basitleştirir ve genel veri artırır. Bu model kullanılarak bir dosya oluşturulur, dosya sistemine yazılır ve ardından kapatılır. Dosya kapatıldıktan sonra içeriğinde değişiklik yapılamaz. Bu, genel sistem performansını artırır ve birçok Büyük Veri uygulaması tarafından gerçekleştirilen görev türleri için iyi çalışır. Dosyanın mevcut içeriği değiştirilemese de HDFS'deki son gelişmeler sonuna yeni veriler eklenmesine olanak tanır. Bu, NoSQL veritabanları için önemli bir gelişmedir çünkü veritabanı günlüklerinin güncellenmesine tanır.
- *Akış erişimi*. Sorguların genellikle birkaç farklı tablodan küçük veri parçaları aldığı işlem işleme sistemlerinin aksine, Büyük Veri uygulamaları genellikle dosyaların tamamını işler. Dosya sistemini tek tek veri öğelerine rastgele erişecek şekilde optimize etmek yerine, Hadoop tüm dosyaların sürekli bir veri akışı olarak toplu işlenmesi için optimize edilmiştir.
- *Hata toleransı*. Hadoop binlerce düşük maliyetli, ortak kullanımlı bilgisayara dağıtılmak üzere tasarlanmıştır. Bu tür binlerce cihazın herhangi bir zamanda bazılarının donanım hataları yaşayacağı varsayılmaktadır. Bu nedenle HDFS, verileri birçok farklı cihaz arasında çoğaltmak üzere tasarlanmıştır, böylece bir cihaz arızalandığında veriler başka bir cihazdan kullanılabilir. Hadoop varsayılan olarak üç replikasyon faktörü kullanır, yani her veri bloğu üç farklı cihazda depolanır. İstenirse her dosya için farklı çoğaltma faktörleri belirlenebilir.

Hadoop çeşitli düğüm türleri kullanır. Bir *düğüm*, sistem içinde bir veya daha fazla türde görev gerçekleştiren bir bilgisayardır. HDFS içinde üç tür düğüm vardır: Şekil 14.4'te gösterildiği gibi istemci düğümü, ad düğümü ve bir veya daha fazla veri düğümü.

Veri düğümleri gerçek dosya verilerini HDFS içinde depolar. HDFS'deki dosyaların bloklar halinde bir araya getirildiğini ve hata toleransı sağlamak için çoğaltıldığını hatırlayın. Sonuç olarak, her blok birden fazla veri düğümünde çoğaltılır. Şekil 14.4'te varsayılan çoğaltma faktörü üç olarak gösterilmektedir, bu nedenle her blok üç veri düğümünde görünür.

Ad düğümü dosya sistemi için meta verileri içerir. Bir HDFS kümesinde genellikle yalnızca bir ad düğümü bulunur. Meta veriler küçük, basit ve kolayca kurtarılabilir olacak şekilde tasarlanmıştır. Meta verilerin küçük tutulması, disk erişimlerini azaltmak ve sistem performansını artırmak için ad düğümünün tüm meta verileri bellekte tutmasını sağlar. Bu önemlidir çünkü yalnızca bir isim düğümü vardır, bu nedenle isim düğümü için çekişme en aza indirilir. Meta veriler öncelikle her dosyanın adından, her dosyayı oluşturan blok numaralarından ve her dosya için istenen çoğaltma faktöründen oluşur. İstemci düğümü, kullanıcı uygulamasını desteklemek için gerektiğinde dosyaları okumak veya yeni dosyalar yazmak için dosya sistemine istekte bulunur.

**Şekil 14.4** Hadoop Dağıtılmış Dosya Sistemi (HDFS)




Bir istemci düğümü yeni bir dosya oluşturmak istediğinde, isim düğümü ile iletişim kurar. İsim düğümü:

- Yeni dosya adını meta veriye ekler.
- Dosya için yeni bir blok numarası belirler.
- Bloğun hangi veri saklanacağına ilişkin bir liste belirler.
- Bu bilgiyi istemci düğümüne geri iletir.

İstemci düğüm, ad düğümü tarafından belirtilen ilk veri düğümüyle bağlantı kurar ve dosyayı bu veri düğümüne yazmaya başlar. Aynı zamanda, istemci düğüm veri düğümüne bloğu çoğaltacak diğer veri düğümlerinin listesini gönderir. Veriler istemci düğümden alındıkça, veri düğümü listedeki bir sonraki veri düğümüyle iletişime geçer ve verileri çoğaltma için bu düğüme göndermeye başlar. Bu ikinci veri düğümü daha sonra listedeki bir sonraki veri düğümüyle bağlantı kurar ve süreç, verilerin bloğu depolayan tüm veri düğümlerine aktarılmasıyla devam eder. İlk blok yazıldıktan sonra, istemci düğümü bir sonraki blok için isim düğümünden başka bir blok numarası ve veri düğümleri listesi alabilir. Tüm dosya yazıldığında, istemci düğümü isim düğümüne dosyanın kapatıldığını bildirir. Veri dosyasının hiçbir zaman gerçekte isim düğümüne iletilmediğine dikkat etmek önemlidir. Bu, sistem performansını yavaşlatabilecek tıkanıklığı önlemek için isim düğümüne veri akışını azaltmaya yardımcı olur.

Benzer şekilde, bir istemci düğümünün bir dosyayı okuması gerekiyorsa, bu dosyayla ilişkili blokların listesini ve bunları tutan veri düğümlerini istemek için ad düğümüyle iletişime geçer. Her bloğun birçok veri düğümünde bulunabileceği göz önüne alındığında, istemci her blok için bloğu ağ üzerinde kendisine en yakın olan veri düğümünden almaya çalışır. Bu bilgiyi kullanarak, istemci düğümü verileri doğrudan bu düğümlerin her birinden okur.

Periyodik olarak, her veri düğümü isim düğümü ile iletişim kurar. Veri düğümleri blok raporları ve kalp atışları gönderir. Bir **blok raporu** her 6 saatte bir gönderilir ve isim düğümüne o veri hangi blokların bulunduğunu bildirir. Kalp atışları her 3 saniyede bir gönderilir. **Kalp atışı**, isim düğümüne veri düğümünün hala kullanılabilir olduğunu bildirmek için kullanılır. Bir veri düğümü; donanım arızası, elektrik kesintisi vb. nedeniyle bir arıza meydana gelirse, ad düğümü bu veri düğümünden bir kalp atışı almayacaktır.

**blok raporu**

Hadoop Dağıtılmış Dosya Sisteminde (HDFS), veri düğümü tarafından her 6 saatte bir isim düğümüne gönderilen ve isim düğümüne hangi blokların o veri düğümünde olduğunu bildiren bir rapor.

**kalp atışı**

Hadoop Dağıtılmış Dosya Sisteminde (HDFS), veri düğümünden isim düğümüne her 3 saniyede bir sinyal gönderilerek isim veri düğümünün hala kullanılabilir olduğu bildirilir.

Sonuç olarak, isim düğümü o veri düğümünü dosyaları okumak ya da yazmak için istemci düğümlerin listelerine dahil etmeyeceğini bilir. Bir veri düğümünden kalp atışı gelmemesi bir bloğun istenen çoğaltma sayısından daha az sayıda çoğaltılmasına neden oluyorsa, isim düğümü "canlı" bir veri düğümünün bloğu başka bir veri düğümünde çoğaltmaya başlamasını sağlayabilir.

Birlikte ele alındığında, HDFS'nin bileşenleri, Büyük Veri uygulamalarının özel işlem gereksinimleri için iyi çalışan güçlü, ancak son derece uzmanlaşmış bir dağıtılmış dosya sistemi üretir. Daha sonra, MapReduce'un HDFS'nin veri depolamasını tamamlamak için nasıl veri işleme sağladığını ele alacağız.

**MapReduce**
Hızlı veri analizi hizmetleri sağlayan açık kaynaklı bir uygulama programlama arayüzü (API); kuruluşların devasa veri depolarını işlemesine olanak tanıyan ana Büyük Veri teknolojilerinden biri.

**haritalama (eşleme)**
Bir MapReduce işinde sıralama ve filtreleme yapan işlev verileri daha büyük bir iş içinde bir alt görev olarak bir anahtar-değer çiftleri kümesine dönüştürür.

**haritalayıcı (eşleyici)**
Bir harita işlevi gerçekleştiren bir program.

**azaltma**
MapReduce işinde, tek bir sonuç üretmek için harita işlevlerinin sonuçlarını toplayan ve özetleyen işlev.

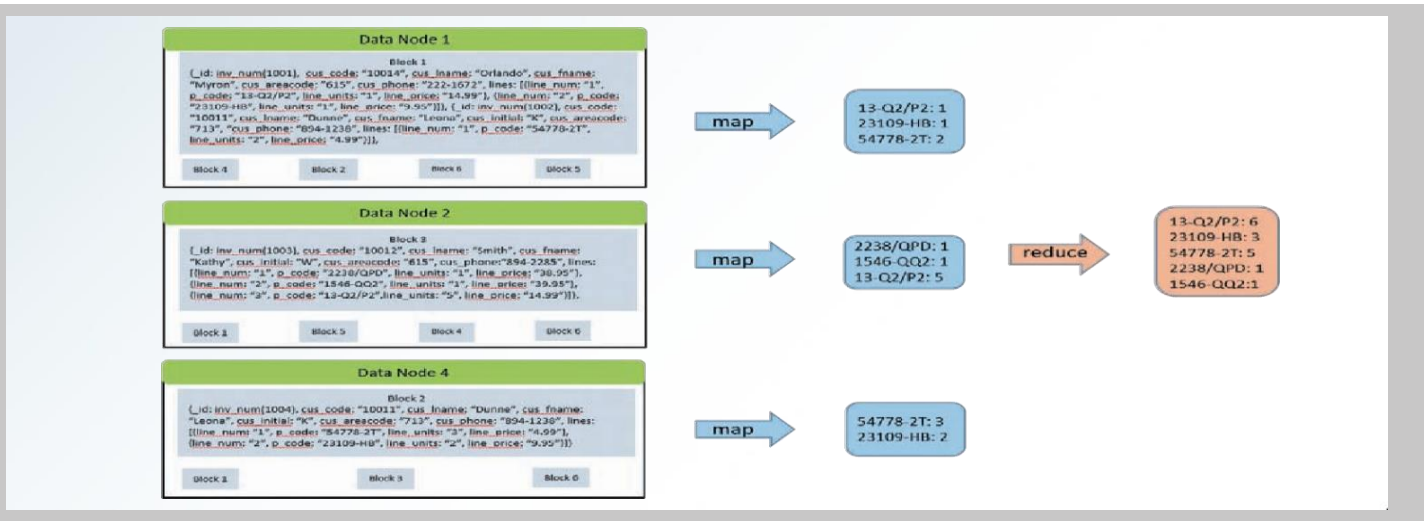
**azaltıcı**
Bir azaltma işlevi gerçekleştiren bir program.

## 14-2b MapReduce

**MapReduce**, kümeler arasında büyük veri kümelerini işlemek için kullanılan bir hesaplama çerçevesidir. Aslında MapReduce'un anlaşılması kolaydır ve *böl ve yönet* prensibini takip eder. MapReduce karmaşık bir görevi alır, bu görevi daha küçük alt görevlere böler, alt görevlerin hepsini aynı anda gerçekleştirir ve ardından orijinal görev için nihai bir sonuç üretmek üzere her bir alt görevin sonucunu birleştirir. Adından da anlaşılacağı gibi, bir eşleme işlevi ile bir azaltma işlevinin birleşimidir. **Haritalama (eşleme)** işlevi bir veri koleksiyonunu alır ve verileri bir anahtar-değer çiftleri kümesi halinde sıralar ve filtreler. Haritalama işlevi, **haritalayıcı (eşleyici)** adı verilen bir program tarafından gerçekleştirilir. **Azaltma** işlevi, hepsi aynı anahtar değerine sahip bir anahtar-değer çiftleri koleksiyonunu alır ve bunları tek bir sonuçta toplar. Azaltma işlevi, **azaltıcı** adı verilen bir program tarafından gerçekleştirilir. Hadoop'un Java tabanlı bir platform olduğunu hatırlayın; bu nedenle eşleme ve azaltma işlevleri ayrıntılı, prosedür odaklı Java programları olarak yazılır.

Şekil 14.5, satılan her bir ürünün toplam birim sayısını belirleyen MapReduce'un basit, kavramsal bir gösterimini sunmaktadır. Şekil 14.5'teki orijinal veriler, fatura numarası anahtar ve fatura verilerinin geri kalanı değer olmak üzere anahtar-değer çiftleri olarak saklanır. Hadoop veri depolama alanındaki verilerin ilişkisel bir veritabanı olmadığını, dolayısıyla verilerin tablolara ayrılmadığını ve her bir gerçeğin yalnızca bir kez depolanmasını sağlayan bir normalleştirme biçimi olmadığını unutmayın. Bu nedenle, orijinal veri deposunda çok sayıda veri kopyası bulunmaktadır. Şekil 14.5'te gösterilen küçük veri alt kümesinde 10011 numaralı müşteri Leona Dunne için bile gereksiz verilerin tutulduğunu unutmayın. Şekilde, eşleme işlevleri her faturayı ayrıştırarak o faturada satılan ürünlerle ilgili verileri bulur. Eşleme işlevinin sonucu, ürün kodunun anahtar ve satır birimlerinin değer olduğu yeni bir anahtar-değer çiftleri listesidir. Azaltma işlevi daha sonra bu anahtar-değer çiftleri listesini alır ve özet sonucu üretmek için her bir anahtarla (ürün kodu) ilişkili değerleri toplayarak bunları birleştirir.

**Şekil 14.5** MapReduce



Daha önce de belirtildiği gibi, Büyük Veri uygulamalarında kullanılan veri setleri son derece büyüktür. Dosyaların tamamının birden fazla düğümden işlenmek üzere merkezi bir düğüme aktarılması muazzam miktarda ağ bant genişliği gerektirecek ve merkezi düğüme inanılmaz bir işlem yükü getirecektir. Bu nedenle, hesaplama programının verileri merkezi bir konumda işlenmek üzere alması yerine, programın kopyaları işlenecek verileri içeren düğümlere "itilir". Programın her kopyası, daha sonra düğümler arasında toplanan ve istemciye geri gönderilen sonuçlar üretir. Bu, HDFS'deki veri dağıtımını yansıtır. Tipik olarak, Hadoop çerçevesi işlenmesi gereken her veri düğümündeki her blok için bir eşleyici dağıtır. Bu, çok fazla sayıda eşleyiciye yol açabilir. Örneğin, 1 TB veri işlenecekse ve HDFS 64 MB blok kullanıyorsa, bu 15.000'den fazla eşleyici programa neden olur. Azaltıcıların sayısı kullanıcı tarafından yapılandırılabilir, ancak en iyi uygulamalar veri düğümü başına yaklaşık bir azaltıcı önermektedir.

## Not

En iyi uygulamalar, belirli bir düğümdeki eşleyici sayısının 100 veya daha az tutulması gerektiğini önermektedir. Ancak, basit eşleme işlevlerine sahip uygulamaların belirli bir düğüm üzerinde 300 kadar eşleyiciyi tatmin edici bir performansla çalıştırdığı durumlar da vardır. Açıkçası, bu durum her bir düğümde mevcut olan bilgi işlem kaynaklarına bağlıdır.

MapReduce uygulaması HDFS'nin yapısını tamamlar, bu da birlikte bu kadar iyi çalışmalarının önemli bir nedenidir. HDFS yapısının bir isim düğümü ve birkaç veri düğümünden gibi, MapReduce da bir **iş izleyici** (programın gerçek adı JobTracker'dır) ve birkaç **görev izleyici** (programlar TaskTrackers olarak adlandırılır) kullanır. İş izleyici, MapReduce işlemleri için merkezi bir kontrol görevi görür ve normalde ad düğümü olarak görev yapan aynı sunucuda bulunur. Görev izleyici programları veri düğümlerinde bulunur. MapReduce çerçevesinin önemli bir özelliği, kullanıcının eşleme ve azaltma işlevleri için Java kodunu yazması ve gönderilen iş için okunacak ve yazılacak girdi çıktı dosyalarını belirtmesi gerektiğidir. Bununla birlikte, iş izleyici verilerin bulunması, hangi düğümlerin kullanılacağının belirlenmesi, işin düğümler için görevlere bölünmesi ve düğümlerin arızalarının yönetilmesi ile ilgilenecektir. Tüm bunlar kullanıcı müdahalesi olmadan otomatik olarak yapılır. Bir kullanıcı bir MapReduce işini işlenmek üzere gönderdiğinde, genel süreç aşağıdaki gibidir:

**iş izleyici**
Bir Hadoop ortamında MapReduce işleme işlerini kabul etmek, dağıtmak, izlemek ve raporlamak için kullanılan merkezi bir kontrol programı.

**görev izleyici**
MapReduce çerçevesindeki bir düğüm üzerinde eşleme ve azaltma görevlerini çalıştırmaktan sorumlu bir program.

1. Bir istemci düğümü (istemci uygulaması) iş izleyiciye bir MapReduce işi gönderir.
2. İş izleyici, hangi veri düğümlerinin bu iş için işlenmesi gereken blokları içerdiğini belirlemek için ad düğümü ile iletişim kurar.
3. İş izleyici, hangi görev izleyicilerin iş için uygun olduğunu belirler. Her görev izleyici belirli sayıda görevi işleyebilir. Unutmayın, Hadoop sisteminde farklı kullanıcılardan birçok MapReduce işi aynı anda çalışıyor olabilir, bu nedenle bir veri düğümü aynı anda farklı işlerden birden fazla eşleyici tarafından işlenmekte olan verileri içerebilir. Bu nedenle, bu düğümdeki görev izleyici, bu yeni istek geldiğinde diğer işler için eşleyicileri çalıştırmakla meşgul olabilir. Veriler birden fazla düğümde çoğaltıldığından, iş izleyici aynı veriler için birden fazla düğüm arasından seçim yapabilir.
4. İş izleyici daha sonra bu düğümlerin her birindeki görev izleyicilerle iletişime geçerek görevin o düğüme ait kısmını tamamlamak üzere eşleyicileri ve indirgeyicileri başlatır.
5. Görev izleyici, eşleme ve azaltma işlevlerini çalıştırmak için yeni bir JVM (Java sanal makinesi) oluşturur. Bu şekilde, bir işlev başarısız olursa veya çökerse, tüm görev izleyici durdurulmaz.
6. Görev izleyici, iş izleyiciye görev izleyicinin hala iş üzerinde çalıştığını (ve düğümlerin daha fazla iş için uygun olduğunu) bildirmek için iş izleyiciye kalp atışı mesajları gönderir.

7. İş izleyici, bir görev yöneticisinin başarısız olup olmadığını belirlemek için kalp atışı mesajlarını izler. Eğer öyleyse, iş takipçisi görevin o kısmını başka bir düğüme yeniden atayabilir.
8. Tüm iş tamamlandığında, iş izleyici işin tamamlandığını belirtmek için durumu değiştirir.
9. İstemci düğüm, iş durumu tamamlanana kadar iş izleyiciyi periyodik olarak sorgular.

**toplu işleme**
Herhangi bir kullanıcı etkileşimi olmadan veri işleme görevlerini baştan sona çalıştıran bir veri işleme yöntemi.

Hadoop sistemi toplu işlemeyi kullanır. **Toplu işleme**, bir programın kullanıcı ile herhangi bir etkileşime girmeden görevi tamamlayarak ya da bir hata ile durarak baştan çalışmasıdır. Toplu işleme genellikle bilgi işlem görevi uzun bir süre veya sistemin işlem kapasitesinin büyük bir bölümünü gerektirdiğinde kullanılır. İşletmeler, sistemlerin genellikle boşta olduğu akşam saatlerinde yıl sonu mali raporlarını çalıştırmak için toplu işlemeyi kullanırken, üniversiteler de öğrenci harç ödemelerinin işlenmesi için toplu işlemeyi kullanabilir. Toplu işleme kötü değildir, ancak sınırlamaları vardır. Sonuç olarak, Hadoop'un daha büyük BT altyapısına entegrasyonunu iyileştirmek için bir dizi tamamlayıcı program geliştirilmiştir. Bir sonraki bölümde bu programlardan bazıları ele alınmaktadır.

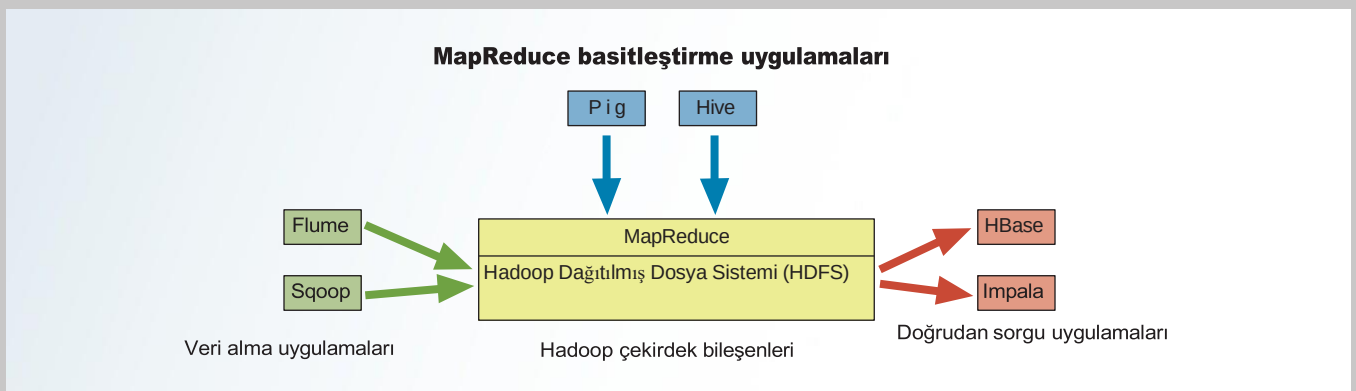
## 14-2c Hadoop Ekosistemi

Hadoop, son derece büyük veri setlerini analiz etme potansiyelinden yararlanan kuruluşlar tarafından yaygın olarak kullanılmaktadır. Ne yazık ki Hadoop oluşturmak, yönetmek ve kullanmak için önemli çaba gerektiren çok düşük seviyeli bir araç olduğundan, oldukça az engelle karşılaşmaktadır. Sonuç olarak, karmaşık Java programlama konusunda yetenekli olmayan kullanıcılar için kullanımı daha kolay ve daha erişilebilir hale getirmeye çalışmak için Hadoop etrafında bir dizi ilgili uygulama ortaya çıkmıştır. Şekil 14.6'da bu tür uygulamalardan bazılarının örnekleri gösterilmektedir. Hadoop kullanan çoğu kuruluş aynı zamanda birbirleriyle etkileşim halinde olan ve birbirlerini tamamlayarak bütün bir uygulama ve araç ekosistemi oluşturan bir dizi başka ilgili ürün de kullanmaktadır. Her ekosistemde olduğu gibi, birbirine bağlı parçalar sürekli olarak gelişmekte ve ilişkileri değişmektedir, bu nedenle oldukça akışkan bir durum söz konusudur. Aşağıda bir Hadoop ekosistemindeki daha popüler bileşenlerden bazıları ve birbirleriyle nasıl ilişkili oldukları açıklanmaktadır.

### MapReduce Basitleştirme Uygulamaları

MapReduce işleri oluşturmak önemli programlama becerileri gerektirir. Eşleyici ve indirgeyici programları daha karmaşık hale geldikçe, beceri gereksinimleri artar ve programları üretmek için gereken süre önemli hale gelir. Bu beceriler çoğu veri kullanıcısının yeteneklerinin ötesindedir.

**Şekil 14.6** Hadoop Ekosisteminden Bir Örnek



Aslında, doğrudan Java'da önemli MapReduce işleri oluşturan kuruluşların sayısı son yıllarda büyük ölçüde azalmıştır ve doğrudan, düşük seviyeli MapReduce programlamasının azalmaya devam edeceği tahmin edilmektedir. Bu nedenle, MapReduce işleri oluşturma sürecini basitleştiren uygulamalar geliştirilmiştir. Bunlardan en popüler ikisi Hive (kovan) ve Pig'dir (domuz).

*Hive (kovan)*, HDFS'nin üzerine oturan bir veri ambarı sistemidir. İlişkisel bir veritabanı değildir, ancak geçici sorgular çalıştırmak için SQL komutlarını taklit eden HiveQL adı verilen kendi SQL benzeri dilini destekler. HiveQL komutları, Hive sorgu motoru tarafından MapReduce işleri kümeleri halinde işlenir. Sonuç olarak, temel işleme toplu iş odaklı olma eğilimindedir ve son derece büyük veri kümeleri üzerinde çok ölçeklenebilir işler üretir. Ancak, işlerin toplu doğası, Hive'ı yalnızca küçük bir veri alt kümesinin çok hızlı bir şekilde döndürülmesini gerektiren işler için kötü bir seçim haline getirir.

*Pig (domuz)*, Pig Latin adlı üst düzey bir komut dosyası dilini Hadoop'ta yürütülmek üzere MapReduce işlerine derlemeye yarayan bir araçtır. Kavram olarak Hive'a benzer, çünkü düşük seviyeli Java programlama yükü olmadan MapReduce işleri üretmenin bir yolunu sağlar. Temel fark, Pig Latince'nin bir komut dosyası dili olması, yani prosedürel olması, HiveQL'in ise SQL gibi bildirimsel olmasıdır. Bildirime dayalı diller, kullanıcının ne istediğini belirtmesine izin verir, nasıl elde değil. Bu, sorgu işleme için çok kullanışlıdır. Prosedürel diller, kullanıcının verilerin nasıl manipüle edileceğini belirtmesini gerektirir. Bu, veri dönüşümleri gerçekleştirmek için çok kullanışlıdır. Sonuç olarak, Pig genellikle verileri bir dizi adımda dönüştüren veri hattı görevleri üretmek için kullanılır. Bu genellikle Bölüm 13'te açıklandığı gibi ETL (çıkarma, dönüştürme ve yükleme) işlemlerinde görülür.

## Veri Alma Uygulamaları

Hadoop'un devasa veri depolama ve veri işleme yeteneklerinden yararlanan kuruluşların karşılaştığı zorluklardan biri, mevcut sistemlerindeki verileri Hadoop kümesine aktarma sorunudur. Bu görevi basitleştirmek için, bu verileri Hadoop'a "almak" veya toplamak için uygulamalar geliştirilmiştir.

*Flume (kanal)*, Hadoop'a veri almak için kullanılan bir bileşendir. Öncelikle web sunucusu günlüklerinden tıklama akışı verileri gibi sunucu günlük dosyalarından büyük veri kümelerini toplamak için tasarlanmıştır. Verileri düzenli bir programa veya belirli olaylara göre içe aktaracak şekilde yapılandırılabilir. Verileri Hadoop'a getirmenin yanı sıra, Flume basit bir sorgu işleme bileşeni içerir, böylece veriler toplanırken veriler üzerinde bazı dönüşümler gerçekleştirme olasılığı vardır. Tipik olarak, Flume verileri HDFS'ye taşır, ancak verileri doğrudan HBase adlı Hadoop ekosisteminin başka bir bileşenine girecek şekilde de yapılandırılabilir.

*Sqoop (şakşak)*, Hadoop ekosistemine daha yeni eklenen bir araçtır. Verileri ilişkisel veritabanı ile HDFS arasında ileri geri dönüştürmek için kullanılan bir araçtır. Sqoop adı (bir kepçe dondurma gibi "scoop (kepçe)" olarak telaffuz edilir) "SQL'den Hadoop'a"nın bir karışımıdır. Konsept olarak Sqoop, verileri HDFS'ye getirmenin bir yolunu sağlaması bakımından Flume'a benzer. Ancak Flume öncelikle günlük dosyalarıyla çalışırken, Sqoop Oracle, MySQL ve SQL Server gibi ilişkisel veritabanlarıyla çalışır. Ayrıca, Flume yalnızca tek yönde çalışırken, Sqoop verileri HDFS'ye ve HDFS'den dışarı olmak üzere her iki yönde de aktarabilir. İlişkisel bir veritabanından HDFS'ye veri aktarırken, veriler her seferinde bir tablo olarak içe aktarılır ve işlem tabloyu satır satır okur. Bu işlem MapReduce kullanılarak yüksek oranda paralelleştirilmiş bir şekilde yapılır, bu nedenle tablonun içeriği genellikle ayrılmış bir biçimde saklanan satırlarla birlikte birkaç dosyaya dağıtılır. Veriler HDFS'ye aktarıldıktan sonra MapReduce işleri veya Hive kullanılarak işlenebilir. Elde edilen veriler daha sonra HDFS'den çoğunlukla geleneksel bir veri ambarı olan ilişkisel veritabanına geri aktarılabilir.

## Doğrudan Sorgu Uygulamaları

Doğrudan sorgu uygulamaları, MapReduce aracılığıyla mümkün olandan daha hızlı sorgu erişimi sağlamaya çalışır. Bu uygulamalar MapReduce işleme katmanından geçmek yerine doğrudan HDFS ile etkileşime girer.

*HBase*, HDFS'nin üzerine oturacak şekilde tasarlanmış sütun yönelimli bir NoSQL veritabanıdır. HBase'in temel özelliklerinden biri yüksek oranda dağıtık olması ve kolayca ölçeklendirilebilecek şekilde tasarlanmış olmasıdır. SQL veya SQL benzeri dilleri desteklemez, bunun yerine etkileşim için Java gibi daha düşük seviyeli dillere güvenir. Sistem MapReduce işlerine dayanmaz, bu nedenle toplu işlemenin neden olduğu gecikmelerden kaçınır, bu da onu verilerin daha küçük alt kümelerini içeren hızlı işleme için daha uygun hale getirir. HBase, seyrek veri kümelerini hızlı bir şekilde işlemede çok iyidir. HBase, Hadoop ekosisteminin en popüler bileşenlerinden biridir ve Facebook tarafından mesajlaşma sistemi için kullanılmaktadır. Sütun yönelimli veritabanları bir sonraki bölümde daha ayrıntılı olarak ele alınacaktır.

*Impala (antilop)*, Hadoop üzerindeki ilk SQL uygulamasıydı. Cloudera tarafından, verileri doğrudan HDFS'den çeken SQL sorgularını destekleyen bir sorgu motoru olarak üretilmiştir. Impala'dan önce, bir kuruluşun Hadoop'taki verileri bir SQL arayüzü aracılığıyla analistlerin kullanımına sunması gerekiyorsa, veriler HDFS'den çıkarılır ve ilişkisel bir veritabanına aktarılırdı. Impala ile analistler, veriler hala HDFS'deyken doğrudan verilere karşı SQL sorguları yazabilirler. Impala, veri düğümlerinde bellek içi önbelleğe almayı yoğun bir şekilde kullanır. Genellikle büyük miktarda veriyi nispeten küçük bir sonuç kümesinde işlemek için uygun bir araç olarak kabul edilir.

**Not**

Impala dışında, bu bölümde açıklanan Hadoop ekosistemi bileşenlerinin her biri Apache Yazılım Vakfı'nın açık kaynaklı, üst düzey projeleridir. Bu projelerin her biri ve diğerleri hakkında daha fazla bilgi www.apache.org adresinde mevcuttur.

## 14-2d Hadoop Geri İttirme

Hadoop Ekosistemi, Büyük Veri ile başa çıkmak için güçlü ve son derece özelleştirilmiş bir çözüm oluşturabilir. Bununla birlikte, eleştirmenleri de yok değil. Ekosisteme modüler yaklaşım, uygulama için önemli bir zorluk yaratmaktadır. Örneğin, MS Word bu tür bir modüler yaklaşım kullanılarak geliştirilmiş olsaydı, bir belge oluşturmak ve düzenlemek için bir programa, belgeyi yazdırmak için başka bir programa, yazım denetimi için başka bir programa, için başka bir programa, grafik veya resim eklemek için başka bir programa ihtiyacınız olurdu. Kelime işlemcinizi tam olarak istediğiniz yeteneklere göre özelleştirebilir ve belgelerinizin nasıl işleneceğini tam olarak özelleştirebilirsiniz. Öte yandan, her biri farklı programcılar tarafından bağımsız olarak geliştirilen bu programların tümünü yalnızca bir dereceye kadar koordinasyonla kurmanızı ve başarıyla entegre etmenizi gerektirir.

Kesin olarak, birçok kuruluş, başka hiçbir çözümün kopyalayamayacağı bir şekilde kendi özel ihtiyaçlarına göre uyarlanmış özelleştirilmiş bir Hadoop ekosistemine sahip olmaktan fayda sağlamaktadır. Ancak, öğrenme eğrisi dik olabilir. IBM ve Cloudera gibi şirketler veri platformları adı verilen kullanıma hazır Hadoop ekosistemleri sunmaktadır. Bunlar bir dizi ekosistem bileşenini kurmak ve entegre etmek için gereken çabayı önemli ölçüde azaltabilirken, platformu kurumunuzun özel ihtiyaçlarına göre uyarlamak için düzinelerce yapılandırma ayarının yapılması ve tüm ekosistemin kurumunuzun iş akışına entegre edilmesi gerekebilir. Aşağıda tartışılan diğer nedenlerin yanı sıra Hadoop'un algılanan karmaşıklıkları, NoSQL veritabanları gibi alternatif çözümlere olan ilginin artmasına yardımcı olmuştur.

# 14-3 NoSQL

**NoSQL**
Geleneksel ilişkisel veritabanı modeline dayanmayan yeni nesil veritabanı yönetim sistemleri.

**NoSQL**, Büyük Veri tarafından temsil edilen zorlukları ele almak için geliştirilen geniş bir dizi ilişkisel olmayan veritabanı teknolojisine verilen talihsiz bir isimdir. Bu isim, NoSQL teknolojilerinin ne olduğunu değil, ne olmadığını tanımlaması bakımından talihsizdir. Aslında bu isim, teknolojilerin ne olmadığını açıklamakta da yetersiz kalmaktadır!

Bu isim, Google, Amazon ve Facebook gibi kuruluşların veri setleri devasa boyutlara ulaştıkça karşılaştıkları sorunlarla başa çıkmak için geliştirdikleri ilişkisel olmayan veritabanı teknolojileri hakkındaki fikirleri tartışmak üzere geliştiricilerden oluşan bir toplantıyı koordine etmeyi kolaylaştırmak için bir Twitter hashtag'i olarak seçildi. *NoSQL* terimi hiçbir zaman bu kategorideki ürünlerin SQL desteği içermemesi gerektiği anlamına gelmemiştir. Aslında, bu tür birçok ürün SQL'i önemli şekillerde taklit eden sorgu dillerini desteklemektedir. Henüz hiç kimse standart SQL'i uygulayan bir NoSQL sistemi üretmemiş olsa da SQL kullanıcılarının geniş tabanı göz önüne alındığında, böyle bir ürün yaratmanın cazibesi açıktır. Son zamanlarda, bazı sektör gözlemcileri NoSQL'in "Sadece SQL Değil" anlamına gelebileceğini söylemeye çalıştılar. Aslında, bir NoSQL ürünü olarak kabul edilme şartı sadece SQL dışındaki dillerin desteklenmesi olsaydı, Oracle, SQL Server, MySQL ve MS Access gibi geleneksel RDBMS ürünlerinin tümü uygun olurdu. Ne olursa olsun, ismin kendisi hakkında endişelenmek yerine terimin atıfta bulunduğu teknolojiler dizisini anlamaya odaklanmanız daha iyi olacaktır.

Geniş anlamda NoSQL terimi altında değerlendirilebilecek yüzlerce ürün bulunmaktadır. Bunların çoğu kabaca dört kategoriden birine uymaktadır: anahtar-değer veri depoları, belge veritabanları, sütun yönelimli veritabanları ve grafik veritabanları. Tablo 14.3'te her türden bazı popüler NoSQL veritabanları gösterilmektedir. Her NoSQL veritabanı açık kaynaklı olarak geliştirilmemiştir, ancak çoğu öyledir.

Sonuç olarak, NoSQL veritabanları genellikle açık kaynak hareketinin bir parçası olarak algılanmaktadır. Buna bağlı olarak, Linux işletim sistemi ile de ilişkilendirilme eğilimindedirler. Maliyet açısından bakıldığında, bir kuruluş on binlerce düğüm içeren bir küme oluşturacaksa, kuruluşun tüm bu düğümler için Windows veya Mac OS lisansları satın almak istememesi mantıklıdır. Tercih, Linux gibi ücretsiz olarak kullanılabilen ve yüksek oranda özelleştirilebilen bir platform kullanmaktır. Bu nedenle, NoSQL ürünlerinin çoğu yalnızca Linux veya Unix ortamında çalışır. Aşağıdaki bölümlerde başlıca NoSQL yaklaşımlarının her biri ele alınmaktadır.

**Tablo 14.3** NoSQL Veritabanları

| NoSQL Kategorisi | Örnek Veritabanları | Geliştirici |
|---|---|---|
| Anahtar-değer veritabanı | Dynamo<br>Riak<br>Redis<br>Voldemort | Amazon<br>Basho<br>Redis Labs<br>LinkedIn |
| Belge veritabanları | MongoDB<br>CouchDB<br>OrientDB<br>RavenDB | MongoDB, Inc.<br>Apache OrientDB<br>Ltd.<br>Hibernating Rhinos |
| Sütun yönelimli veritabanları | HBase Cassandra<br>Hypertable | Apache<br>Apache (başlangıçta Facebook)<br>Hypertable, Inc. |
| Grafik veritabanları | Neo4j<br>ArangoDB<br>GraphBase | Neo4j<br>ArangoDB, LLC<br>FactNexus |

## 14-3a Anahtar-Değer Veritabanları

**Anahtar-değer (key-value, KV) veritabanları** kavramsal olarak NoSQL veri modellerinin en basitidir. KV veritabanı, verileri anahtar-değer çiftleri koleksiyonu olarak depolayan bir NoSQL veritabanıdır. Anahtar, değer için bir tanımlayıcı görür. Değer metin, XML belgesi ya da resim gibi herhangi bir şey olabilir. Veritabanı, değer bileşeninin içeriğini veya anlamını anlamaya çalışmaz; veritabanı sadece anahtar için sağlanan değeri depolar. Değer bileşenindeki verilerin anlamını anlamak, verileri kullanan uygulamaların işidir. Yabancı anahtarlar yoktur; aslında anahtarlar arasında hiçbir ilişki i izlenemez. Bu, DBMS'nin gerçekleştirmesi gereken işi basitleştirerek KV veritabanlarını temel işlemler için son derece hızlı ve ölçeklenebilir hale getirir.

**Anahtar-değer (key-value, KV) veritabanı**
Verileri bir anahtar- değer koleksiyonu olarak depolayan bir NoSQL veritabanı modeli değer bileşeninin DBMS tarafından anlaşılamadığı çiftler.

**kova**
Bir anahtar-değer veritabanında, ilgili anahtar- değer çiftlerinden oluşan mantıksal bir koleksiyon.

Anahtar-değer çiftleri tipik olarak "kovalar" halinde düzenlenir. Bir **kova** kabaca bir tablonun KV veritabanı eşdeğeri olarak düşünülebilir. Bir kova, anahtarların mantıksal bir gruplamasıdır. Anahtar değerler bir kova içinde benzersiz olmalıdır, ancak kovalar arasında çoğaltılabilirler. Tüm veri işlemleri kova artı anahtara dayalıdır. Başka bir deyişle, anahtar-değer çiftinin değer bileşenindeki herhangi bir şeye dayalı olarak verileri sorgulamak mümkün değildir. Tüm sorgular kova ve anahtar belirtilerek gerçekleştirilir.

KV veritabanları üzerindeki işlemler oldukça basittir; yalnızca *get*, *store* ve *delete* işlemleri kullanılır. *Get* veya *fetch*, çiftin değer bileşenini almak için kullanılır. *Store*, bir değeri bir anahtara yerleştirmek için kullanılır. Kova 1 anahtar kombinasyonu mevcut değilse, yeni bir anahtar-değer çifti olarak eklenir. Kova 1 anahtar kombinasyonu mevcutsa, mevcut değer bileşeni yeni değerle değiştirilir. *Delete*, bir anahtar-değer çiftini kaldırmak için kullanılır. Şekil 14.7'de üç anahtar-değer çiftine sahip bir müşteri kovası gösterilmektedir. KV modeli değer bileşenindeki verilere dayalı sorgulara izin vermediğinden, örneğin müşteri soyadına dayalı bir anahtar-değer çifti için sorgu yapmak mümkün değildir. Aslında KV DBMS, değer bileşeninin içeriğini anlamadığı için müşteri soyadı diye bir şeyin varlığından bile haberdar değildir. Bir uygulama, KV DBMS'nin kova müşteri ve anahtar 10011 için anahtar-değer çiftini döndürmesini sağlamak için bir *get* komutu verebilir, ancak müşterinin soyadını, adını ve diğer özelliklerini bulmak için değer bileşenini nasıl bilmek uygulamaya bağlıdır. (Şekil 14.7 ile ilgili önemli bir not: Şekilde anahtar-değer çiftleri sekme biçiminde görünse de sekme biçiminin bileşenleri görsel olarak ayırt etmeye yardımcı olmak için sadece bir kolaylık olduğunu unutmayın. Gerçek anahtar-değer çiftleri tablo benzeri bir yapıda saklanmaz).

## 14-3b Belge Veritabanları

**Belge veritabanları** kavramsal olarak anahtar-değer veritabanlarına benzer ve neredeyse KV veritabanlarının bir alt türü olarak kabul edilebilirler. Bir belge veritabanı, verileri anahtar-değer çiftleri halinde etiketlenmiş belgelerde depolayan bir NoSQL veritabanıdır. Değer bileşeninin herhangi bir veri türünü içerebileceği bir KV veritabanının aksine, bir belge veritabanı değer bileşeninde her zaman bir belge saklar. Belge, XML, **JSON (JavaScript Object Notation)** veya **BSON (Binary JSON)** gibi herhangi bir kodlanmış formatta olabilir. Bir diğer önemli fark ise KV veritabanları değer bileşeninin içeriğini anlamaya çalışmazken, belge veritabanlarının bunu yapmasıdır. Etiketler bir belgenin adlandırılmış bölümleridir. Örneğin, bir belge, belgedeki hangi metnin belgenin başlığını, yazarını ve gövdesini temsil ettiğini tanımlamak için etiketlere sahip olabilir. Belgenin gövdesi içinde, bölümleri ve kısımları belirtmek için ek etiketler olabilir. Belgelerde etiketlerin kullanılmasına rağmen, belge veritabanları şemasız olarak kabul edilir, yani depolanan verilere önceden tanımlanmış bir yapı empoze etmezler. Bir belge veritabanı için şemasız olmak, tüm belgelerin etiketleri olmasına rağmen, tüm belgelerin aynı etiketlere sahip olması gerekmediği, dolayısıyla her belgenin kendi yapısına sahip olabileceği anlamına gelir. Bir belge veritabanındaki etiketler son derece önemlidir çünkü belge veritabanlarının KV veritabanlarına göre sahip olduğu ek yeteneklerin çoğunun temelini oluştururlar. Belge içindeki etiketlere DBMS tarafından erişilebilir, bu da karmaşık sorgulamayı mümkün kılar.

**belge veritabanı**
Verileri, değer bileşeninin etiket kodlu bir belgeden oluştuğu anahtar-değer çiftleri halinde depolayan bir NoSQL veritabanı modeli

**JSON (JavaScript Nesne Gösterimi)**
Bir belgedeki nitelikleri ve değerleri tanımlayan, veri alışverişi için insan tarafından okunabilir bir metin formatı.

**BSON (İkili JSON)**
İkili nesneler de dahil olmak üzere ek veri türlerini içerecek şekilde JSON formatını genişleten veri alışverişi için bilgisayar tarafından okunabilir bir format.

**Şekil 14.7** Anahtar Değer Veritabanı Depolama

**Kova = Müşteri**

| Anahtar | Değer |
|---|---|
| 10010 | {LName: "Ramas", FName: "Alfred", Initial: "A" Areacode: "615", Phone: "844-2573", Balance: "0"} |
| 10011 | {LName: "Dunne", FName: "Leona", Initial: "K", Areacode: "713", Phone: "894-1238", Balance: "0"} |
| 10014 | {LName: "Orlando", FName: "Myron", Areacode: "615", Phone: "222-1672", Balance: "0"} |

**Şekil 14.8** Belge Veritabanı Etiketli Format

**Tahsilat = Müşteri**

| Anahtar | Belge |
|---|---|
| 10010 | {LName: "Ramas", FName: "Alfred", Initial: "A", Areacode: "615", Phone: "844-2573", Balance: "0"} |
| 10011 | {LName: "Dunne", FName: "Leona", Initial: "K", Areacode: "713", Phone: "894-1238", Balance: "0"} |
| 10014 | {LName: "Orlando", FName: "Myron", Areacode: "615", Phone: "222-1672", Balance: "0"} |
| | |

KV veritabanlarının anahtar-değer çiftlerini kova adı verilen mantıksal gruplar halinde gruplandırması gibi, belge veritabanları da belgeleri **koleksiyon** adı verilen mantıksal gruplar halinde gruplandırır. Bir belge koleksiyon ve anahtar belirtilerek alınabilirken, etiketlerin içeriğine göre sorgulama yapmak da mümkündür. Örneğin, Şekil 14.8, Şekil 14.7'deki aynı verileri, ancak bir belge veritabanı için etiketlenmiş bir biçimde temsil etmektedir. DBMS belgelerdeki etiketlerin farkında olduğundan, Balance etiketinin 0 değerine sahip olduğu tüm belgeleri alan sorgular yazmak mümkündür. Belge veritabanları, sorgularda bakiyelerin toplanması veya ortalamasının alınması gibi bazı toplu işlevleri bile destekler. Bu bölümün ilerleyen kısımlarında MongoDB belge veritabanındaki bazı temel işlemleri öğreneceksiniz ve Ek P, MongoDB ile Çalışma, uygulamalı bir öğretici içerir.

**koleksiyon**
Belge veritabanlarında, mantıksal bir depolama birimi kabaca ilişkisel bir veritabanındaki bir tabloya benzer şekilde benzer belgeler içerir.

Belge veritabanları, bir belgenin belirli bir konuyla ilgili verilerin bir parçası değil, tamamen kendi kendine yeten bir belge olduğu varsayımıyla çalışma eğilimindedir. İlişkisel veritabanları, iş ortamındaki karmaşık verileri bir dizi ilgili tabloya ayrıştırır. Örneğin, siparişlerle ilgili veriler müşteri, fatura, hat ve ürün tablolarına ayrıştırılabilir. Bir belge veritabanı, bir siparişle ilgili tüm verilerin tek bir sipariş belgesinde olmasını bekler. Bu nedenle, Siparişler koleksiyonundaki her bir sipariş belgesi, müşteri, siparişin kendisi ve bu siparişte satın alınan ürünler hakkındaki verileri tek bir bağımsız belge olarak içerecektir. Belge veritabanları, ilişkisel modelde algılandığı gibi ilişkileri saklamaz ve genellikle birleştirme işlemleri için destekleri yoktur.

## 14-3c Sütun Yönelimli Veritabanlar

"*Sütun yönelimli veritabanı*" terimi, genellikle birbiriyle karıştırılan iki farklı teknoloji kümesini ifade edebilir. Bir anlamda, sütun odaklı veritabanı veya sütunlu veritabanı, **satır merkezli depolama** yerine **sütun merkezli depolama** kullanan geleneksel, ilişkisel veritabanı teknolojilerini ifade edebilir. İlişkisel veritabanları verileri mantıksal tablolar halinde sunar; ancak veriler aslında veri satırlarını içeren veri bloklarında saklanır. Belirli bir satıra ait tüm veriler, aynı veri bloğundaki birçok satırla birlikte sırayla depolanır. Bir tabloda çok sayıda veri satırı varsa, satırlar çok sayıda veri bloğuna yayılacaktır. Şekil 14.9, fiziksel olarak beş veri bloğunda depolanan 10 veri satırına sahip bir ilişkisel tabloyu göstermektedir. Satır merkezli depolama, bir veri satırını almak için gereken disk okuma sayısını en aza indirir. Bir veri satırını almak için Şekil 14.9'da gösterildiği gibi yalnızca bir veri bloğuna erişmek gerekir.

**sütun merkezli depolama**
Verilerin bloklar depolandığı ve birçok satırda tek bir sütundaki verileri tutan fiziksel bir veri depolama tekniği.

**satır merkezli depolama**
Verilerin, belirli bir satır kümesinin tüm sütunlarındaki verileri tutan bloklar halinde depolandığı fiziksel bir veri depolama tekniği.

Unutmayın, işlemsel sistemlerde normalleştirme, fazlalığı azaltmak ve küçük veri kümelerinin hızlı manipülasyon hızını artırmak için karmaşık verileri ilgili tablolara için kullanılır. Bu manipülasyonlar satır odaklı olma eğilimindedir, bu nedenle satır odaklı depolama çok iyi çalışır. Ancak, büyük bir satır kümesinde küçük bir sütun kümesini alan sorgularda, çok sayıda disk erişimi gerekir. Örneğin, her müşterinin yalnızca şehrini ve eyaletini almak isteyen bir sorgu, bu verileri almak için bir müşteri satırı içeren her veri bloğuna erişmek zorunda kalacaktır. Şekil 14.9'da bu, her müşterinin şehir ve eyaletini almak için beş veri bloğuna erişmek anlamına gelir.