KOCAELİ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

DAĞITIK SİSTEMLERE GİRİŞ

APACHE KAFKA

MUHAMMED ŞARA 150201123
MUNİR KARSLI 150201134
HOUSSEM MENHOUR 140201115
JASMIN SUKURICA 140201117

KOCAELİ 2019

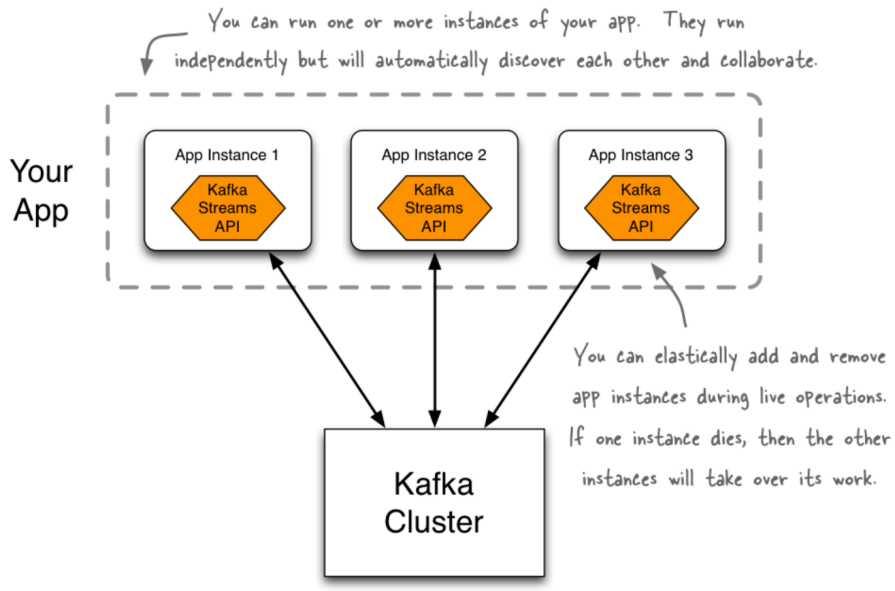
İÇİNDEKİLER

ŞEKİL TABLOSU

1. TANIM

Bir Java kütüphanesi üzerinden erişilebilen Apache Kafka'nın Akış API'si, yüksek oranda ölçeklenebilir, elastik, hataya dayanıklı, dağıtılmış uygulamalar ve mikro hizmetler oluşturmak için kullanılabilir. Birincisi ve en önemlisi, Kafka Streams API, temel işinize güç veren gerçek zamanlı uygulamalar oluşturmanıza olanak sağlar. Kafka'da depolanan verileri işlemek için en kolay ve en güçlü teknolojidir. Uygulama durumunun etkin yönetimi, hızlı ve verimli toplanmalar ve birleşimler gibi akış işleme için önemli kavramlar üzerine kurulur, olay zamanı ile işlem zamanı arasında doğru bir şekilde ayrım yapar ve geçmiş ve sıra dışı verilerin sorunsuz bir şekilde ele alınmasını sağlar.

Kafka Streams API'sının benzersiz bir özelliği, onunla kurduğunuz uygulamaların normal Java uygulamaları olmasıdır. Şekil 1' de de görüldüğü gibi bu uygulamalar diğer Java uygulamalarında olduğu şekilde paketlenebilir, konuşlandırılabilir ve izlenebilir - ayrı işleme kümeleri veya benzer özel amaçlı ve pahalı altyapı kurmaya gerek yoktur [1].



Şekil 1. Kafka Stream API [1].

Şekil 1'de Kafka Streams API'sini kullanan bir uygulama Java örneği. Diğer Java uygulamaları için yaptığı gibi paketlenir, dağıtılır ve takip edilir. Buna rağmen, uygulamalar yüksek derecede ölçeklenebilir, elastik ve hataya dayanıklı olacaktır [1]

Bir akış platformunun üç temel özelliği vardır:

- Mesaj kuyruğuna veya kurumsal mesajlaşma sistemine benzeyen kayıt akışlarını yayınlayan ve abone olan.
- Kayıt akışlarını hataya dayanıklı ve dayanıklı bir şekilde saklayan.
- Kayıt akışlarını oluştukça işleyen.

Kafka genellikle iki geniş uygulama sınıfı için kullanılır:

- Sistemler veya uygulamalar arasında güvenilir şekilde veri alan gerçek zamanlı akışlı veri hatları oluşturmak
- Veri akışlarını dönüştüren veya bunlara tepki veren gerçek zamanlı akış uygulamaları oluşturma

Kafka hakkında önemli durumlar:

- Kafka, birden fazla veri merkezine yayılabilen bir veya daha fazla sunucuda küme olarak çalıştırılır.
- Kafka kümesi, konu adında kategorilerde kayıt akışlarını saklar.
- Her kayıt bir anahtar, bir değer ve bir zaman damgasından oluşur.

1.1. Örnek Kullanım Olayları

Kafka Streams API, çok çeşitli kullanım durumlarına ve endüstrilere uygulanabilir.

- Seyahat şirketleri, bireysel müşteriler için en uygun fiyatı bulmak, ek hizmetleri çapraz satmak ve rezervasyon ve rezervasyonları işlemek için gerçek zamanlı kararlar almak üzere Kafka Streams API ile uygulamalar yapabilir.

- Finans endüstrisi, potansiyel risklerin gerçek zamanlı görünümleri için veri kaynaklarını toplamak ve sahte işlemleri tespit etmek ve en aza indirmek için uygulamalar oluşturabilir.

- Lojistik şirketleri, gönderilerini hızlı, güvenilir ve gerçek zamanlı olarak takip etmek için uygulamalar yapabilir.

- Perakendeciler bir sonraki en iyi teklifler, kişiselleştirilmiş promosyonlar, fiyatlandırma ve envanter yönetimi konularında gerçek zamanlı karar vermek için uygulamalar geliştirebilirler.

- Otomotiv ve üretim şirketleri, üretim hatlarının optimum performans göstermesini sağlamak, tedarik zincirlerine dair gerçek zamanlı bilgiler edinmek ve bir denetimin gerekip gerekmediğine karar vermek için bağlı araçlardan telemetri verilerini izlemek için uygulamalar yapabilir.

- Ve daha fazlası.

1.2. Kafka'nın dört temel API'si vardır:

1.2.1 Producer API

Bir uygulamanın bir veya daha fazla Kafka konusuna bir kayıt akışı yayınlamasına izin verir [2].

1.2.2 Consumer API

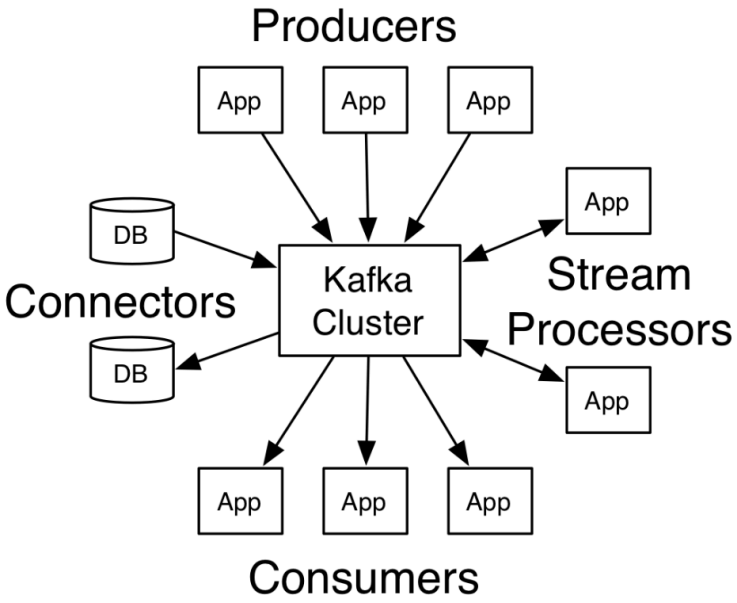
Bir uygulamanın bir veya daha fazla konuya abone olmasını ve kendilerine üretilen kayıt akışını işlemesini sağlar [2].

1.2.3 Stream API

Bir uygulamanın bir akış işlemcisi olarak hareket etmesine, bir veya daha fazla konudan bir girdi akışını tüketmesine ve bir veya daha fazla çıkış konularına bir çıkış akışı oluşturmasına izin vererek giriş akışlarını çıkış akışlarına etkili bir şekilde dönüştürür [2].

1.2.4 Connector API

Kafka konularını mevcut uygulamalara veya veri sistemlerine bağlayan yeniden kullanılabilir üreticiler veya tüketiciler oluşturmaya ve çalıştırmaya izin verir. Örneğin, ilişkisel bir veritabanına bir bağlayıcı, tablodaki her değişikliği yakalayabilir [2].



Şekil 2. Kafka API [2]

Şekil 2 de ki gibi Kafka'da müşteriler ve sunucular arasındaki iletişim, basit, yüksek performanslı, dilde bir TCP protokolü ile yapılır. Bu protokol sürümlendirilmiştir ve eski sürümle geriye dönük uyumluluğu korur.

2. KAFKA ÖZELLİKLERİ

2.1. Dağıtık

Kütüğün bölümleri, Kafka kümesindeki sunucular üzerinde dağıtılır ve her bir sunucunun verileri yönetir ve bölümlerin paylaşılmasını talep eder. Her bölüm, hata toleransı için yapılandırılabilir bir dizi sunucu arasında çoğaltılır.

Her bölüm "lider" olarak çalışan bir sunucuya ve "takipçi" olarak çalışan sıfır ya da daha fazla sunucuya sahiptir. Lider, bölüme ilişkin tüm okuma ve yazma isteklerini yerine getirirken takipçiler lideri pasif olarak kopyalarlar. Lider başarısız olursa, takipçilerden biri otomatik olarak yeni lider olacak. Her sunucu bazı bölümleri için lider ve diğerleri için takipçisi olarak hareket eder, böylece yük küme içinde iyi dengelenir.

2.2. Coğrafi Çoğaltma

Kafka MirrorMaker, kümeleriniz için coğrafi çoğaltma desteği sağlar. MirrorMaker ile, mesajlar birden fazla veri merkezlerinde veya bulut bölgelerinde çoğaltılır. Bunu yedekleme / kurtarma için aktif / pasif senaryolarda kullanabilirsiniz; veya verileri kullanıcılarınıza daha yakın bir yere yerleştirmek veya veri konum gereksinimlerini desteklemek için aktif / aktif senaryolarda.

2.3. Çoklu Kullanım Hakkı

Kafka'yı çoklu kullanım hakkı bir çözüm olarak dağıtabilirsiniz. Çoklu kullanım hakkı, hangi konuların veri üretebileceğini veya tüketebileceğini yapılandırarak etkinleştirilir. Kotalar için operasyon desteği de var. Yöneticiler, istemciler tarafından kullanılan aracı kaynakları kontrol etme istekleri üzerine kota belirleyebilir ve uygulayabilir.

2.4. Güvence

Bir üretici tarafından belirli bir konu bölümüne gönderilen mesajlar, gönderildikleri sıraya eklenir. Yani, M1 kaydı aynı üretici tarafından M2 kaydıyla gönderilirse ve önce M1 gönderilirse, M1 M2'den daha düşük bir kaymaya sahip olacak ve kütükte daha erken görünecektir.
Tüketici örneği, kayıtları günlükte depolanma sırasına göre görür.
N çoğaltma faktörü olan bir konu için, günlüğe kaydedilmiş herhangi bir kaydı kaybetmeden N-1 sunucu arızalarına kadar tolerans gösterir.

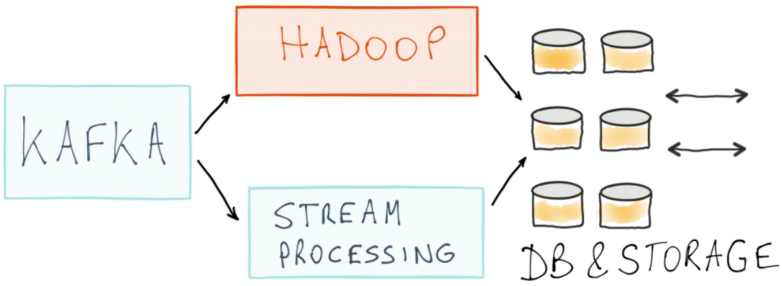
2.5. Depolama

Bunların tüketilmesinden ayrılan mesajların yayınlanmasına izin veren herhangi bir mesaj kuyruğu, uçuş içi mesajlar için bir depolama sistemi görevi görmektedir. Kafka hakkında farklı olan, çok iyi bir depolama sistemi olmasıdır.

Şekil 3' de görüldüğü gibi kafka'ya yazılan veriler diske yazılır ve hata toleransı için çoğaltılır. Kafka, üreticilerin bir onay beklemelerine izin verir, böylece bir yazı tamamen kopyalanıncaya kadar tam olarak kabul edilmez ve sunucu başarısız olsa bile devam etmesi garanti edilir.

Kafka'nın iyi kullandığı disk yapıları iyi ölçeklendirilir - Kafka, sunucuda 50 KB veya 50 TB tutarında kalıcı veri olsa da aynı işlemi gerçekleştirir.

Depolamayı ciddiye almanın ve müşterilerin okuma pozisyonlarını kontrol etmelerine izin vermenin bir sonucu olarak, Kafka'yı yüksek performanslı, düşük gecikmeli işleme günlük depolama, çoğaltma ve yaymaya adanmış bir tür özel amaçlı dağıtılmış dosya sistemi olarak düşünülebilir [2] .



Şekil 3. Kafka Depolama Sistemi [2]

2.6. Parçaları bir araya getirmek

Mesajlaşma, saklama ve aktarma işlemlerinin bu kombinasyonu olağandışı görünebilir ancak Kafka'nın aktarma platformu olarak oynadığı rol için önemlidir.

HDFS gibi dağıtılmış bir dosya sistemi, toplu işlem için statik dosyaların depolanmasına izin verir. Etkili böyle bir sistem geçmişe ait geçmiş verilerin depolanmasına ve işlenmesine olanak tanır.

Geleneksel bir kurumsal mesajlaşma sistemi, abone olduktan sonra gelecek olan gelecekteki mesajları işlemenizi sağlar. Bu şekilde oluşturulmuş uygulamalar geldiğinde verileri işler.

Kafka, bu yeteneklerin her ikisini de bir araya getirir ve kombinasyon, hem akışkanlık uygulamaları için hem de akışkan veri boru hatları için bir platform olarak Kafka kullanımı için kritiktir.

Depolama ve düşük gecikmeli abonelikleri birleştirerek, akış uygulamaları hem geçmiş hem de gelecekteki verileri aynı şekilde ele alabilir. Bu, geçmiş ve depolanan verileri işleyebilen tek bir uygulamadır, ancak en son kayda ulaştığında sona ermek yerine gelecekteki veriler geldikçe işleme devam edebilir. Bu, mesajla çalışan uygulamaların yanı sıra toplu işlemeyi de içeren genelleştirilmiş bir akış işleme kavramıdır.

Aynı şekilde veri aktarımı boru hatları için de gerçek zamanlı etkinliklere abonelik kombinasyonu, Kafka'nın çok düşük gecikmeli boru hatları için kullanılmasını mümkün kılıyor; Ancak, verilerin güvenilir bir şekilde saklanması, verilerin teslim edilmesinin garanti edilmesi gereken kritik veriler için veya sadece periyodik olarak veri yükleyen veya bakım için uzun süre düşebilecek çevrimdışı sistemler ile entegrasyon için kullanılmasını mümkün kılar. Akış işleme tesisleri, verileri geldikçe dönüştürmeyi mümkün kılar.

3. MESAJLAŞMA SİSTEMİ NEDİR?

Bir uygulamadan diğerine veri transferinden bir Mesajlaşma Sistemi sorumludur, bu nedenle uygulamalar verilere odaklanabilir, ancak nasıl paylaşılacağı konusunda endişelenmeyin. Dağıtılmış mesajlaşma, güvenilir mesaj sıralaması kavramına dayanır. Mesajlar, istemci uygulamaları ve mesajlaşma sistemleri arasında senkronize olmayan bir şekilde sıraya alınmaz. İki tür mesajlaşma modeli mevcuttur - biri noktadan noktaya diğeri yayınlamak - abone olmak (pub-sub) mesajlaşma sistemidir. Mesajlaşma modellerinin çoğu pub-sub'u takip eder.

3.1. Mesajlaşma Sistemi Olarak Kafka

Geleneksel olarak mesajlaşmanın iki modeli vardır: sıraya koyma ve yayınlama-abone olma. Kuyrukta, bir tüketici havuzu bir sunucudan okuyabilir ve her kayıt bunlardan birine gider; Yayın-abone olarak kayıt tüm tüketicilere yayınlanmaktadır. Bu iki modelin her birinin gücü ve zayıflığı vardır. Kuyruğa almanın gücü, işlemlerinizi ölçeklendirmenize izin veren birden fazla tüketici örneğine göre verilerin işlenmesini bölmenize izin vermesidir. Ne yazık ki, sıralar çok abone değil bir işlem gittiği verileri okuduğunda. Yayınla-abone ol, verileri birden fazla işleme yayınlamanıza izin verir, ancak her mesaj her aboneye gittiğinden, ölçekleme işlemine sahip değildir.

Kafka'daki tüketici grubu kavramı bu iki kavramı genelleştirir. Bir kuyrukta olduğu gibi, tüketici grubu, bir işlem topluluğuna (tüketici grubu üyeleri) işlemeyi bölmenize izin verir. Yayınla-abone ol, Kafka, birden fazla tüketici grubuna mesaj yayınlamana izin verir.

Kafka'nın modelinin avantajı, her konunun bu özelliklere sahip olması - işlemeyi ölçeklendirebilir ve aynı zamanda çoklu abone olması - birini veya diğerini seçmeye gerek kalmamasıdır.

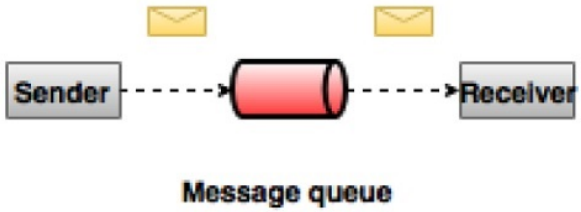
Kafka, geleneksel bir mesajlaşma sisteminden daha güçlü sipariş garantilerine sahiptir.

Geleneksel bir sıra, sunucudaki kayıtları sırayla tutar ve birden fazla tüketici sıradan tüketirse, sunucu kayıtları depolandıkları sırayla dağıtır. Bununla birlikte, sunucu kayıtları düzenli bir şekilde dağıtmasına rağmen, kayıtlar tüketicilere eşzamansız olarak gönderilir, bu nedenle farklı tüketicilere sıra dışı gelebilirler. Bu, kayıtların sıralanmasının paralel tüketimin varlığında kaybolduğu anlamına gelir.
Mesajlaşma sistemleri çoğu zaman bu sorunu giderir, yalnızca bir işlemin bir sıradan tüketilmesine izin veren "özel tüketici" kavramıyla çalışır, ancak bu elbette işleme sürecinde paralellik olmadığı anlamına gelir.

Kafka daha iyi yapar. Kafka, paralellik nosyonuna (bölüme) sahip olan Kafka, bir tüketici süreçleri havuzu için hem sipariş garantisi hem de yük dengelemesi sağlayabilir. Bu, konudaki bölümleri tüketici grubundaki tüketicilere tahsis ederek elde edilir, böylece her bölüm gruptaki tek bir tüketici tarafından tüketilir. Bunu yaparak, tüketicinin bu bölümün tek okuyucusu olmasını ve verileri sırayla tüketmesini sağlıyoruz. Birçok bölüm olduğundan, bu hala birçok tüketici örneğindeki yükü dengeler. Bununla birlikte, bir tüketici grubunda bölümlerden daha fazla tüketici örneği bulunmadığı unutulmamalıdır.

3.1.1. Noktadan Noktaya Mesajlaşma Sistemi

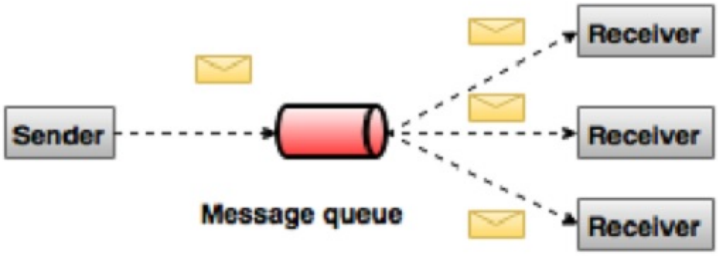
Noktadan noktaya bir sistemde, iletiler kuyrukta devam eder. Şekil 4'te görüldüğü gibi bir veya daha fazla tüketici sıradaki mesajları tüketebilir, ancak belirli bir mesaj yalnızca en fazla bir tüketici tarafından tüketilebilir. Tüketici sıradaki bir iletiyi okuduğunda, bu sıradan kaybolur. Bu sistemin tipik örneği, her bir siparişin bir Sipariş İşlemcisi tarafından işleneceği bir Sipariş İşleme Sistemi'dir, ancak Birden Çok Sipariş İşlemcisi aynı anda çalışabilir. Aşağıdaki diyagram yapıyı göstermektedir [3].



Şekil 4. Noktadan Noktaya İletim [3]

3.1.2. Yayınla-Abone Ol Mesajlaşma Sistemi

Yayınla-abone ol sisteminde, bir konudaki mesajlara devam edilir. Şekil 5'de görüldüğü gibi noktadan noktaya sistemin aksine, tüketiciler bir veya daha fazla konuya abone olabilir ve bu konudaki tüm mesajları kullanabilirler Şekil 5'de bu durum görsel hali bulunmaktadır. Yayın-Abone sisteminde, mesaj üreticilerine yayıncılar, mesaj tüketicilerine abone denir. Gerçek hayattan bir örnek, spor, sinema, müzik vb. Farklı kanalları yayınlayan Dish TV'dir ve herkes kendi kanallarına abone olabilir ve abone olunan kanalları olduğunda bunları alabilir [3]

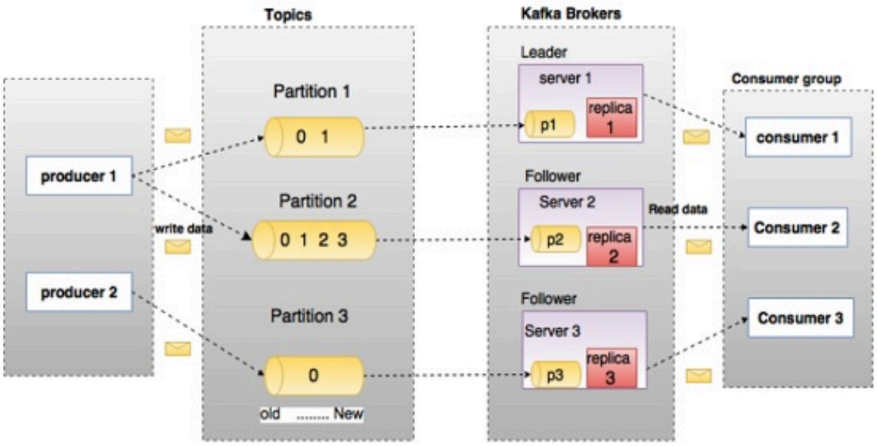


Şekil 5. Yayınla-Abone Ol ile İletişim [3]

4. KAFKA MİMARİSİ

4.1. Üreticiler

Üreticiler, seçtikleri konulara verilerini mesaj olarak yayınlarlar. Üretici, hangi bölüme konu içerisinde hangi kayıt atanacağını seçmekle sorumludur. Şekil 6 da görüldüğü üzere üreticiler, konulara (topic) mesaj gönderirler. Bu mesajlar Partiton denilen parçalardan oluşmaktadır.

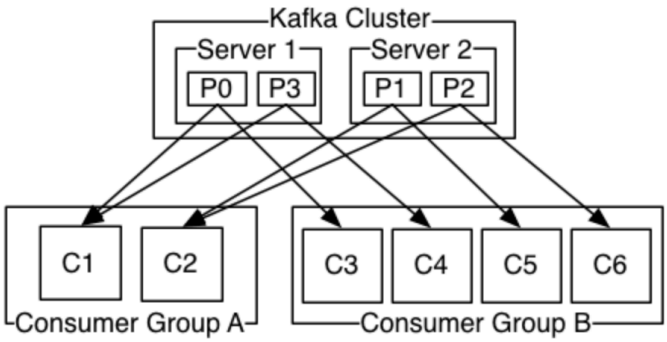


Şekil 6. Kafka Mimarisi Ayrıntılı Şema Gösterimi [3]

4.2. Tüketiciler

Tüketiciler kendilerini bir consumer grubu adıyla etiketler ve bir konuya yayınlanan her mesajı, abone olan her tüketici grubun içindeki bütün mesajlara ulaşabilir. Tüketiciler aynı makinede veya farklı makinelerde olabilirler. Şekil 7'de gösterildiği gibi tüm tüketici örnekleri aynı tüketici grubuna sahipse, kayıtlar tüketici örnekleri üzerinde etkili bir şekilde yük dengelenir.

Tüm tüketici örnekleri farklı tüketici gruplarına sahipse, her kayıt tüm tüketici işlemlerine yayınlanacaktır.



Şekil 7. Tüketici Sunucu İstek Yapısı [2]

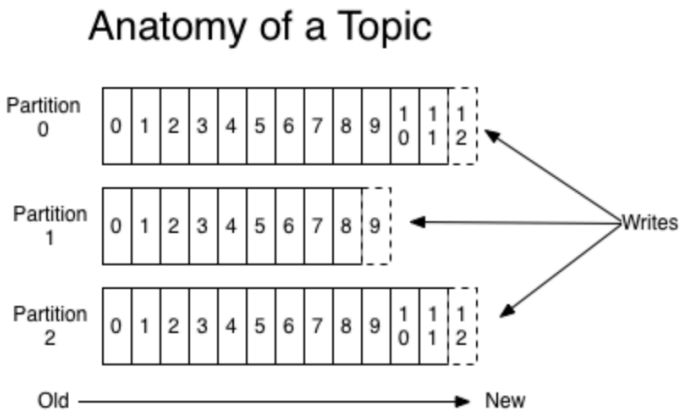
Şekil 7’de iki tüketici grubu ile dört bölüme (P0-P3) ev sahipliği yapan iki sunucu Kafka kümesi görülmektedir. Tüketici grubu A'nın iki tüketici örneği var ve B grubunun dört tane tüketici örneği var.

Kafka'da tüketimin uygulanma yolu, logdaki bölümleri tüketici örnekleri üzerinden bölüştürmektir, böylece her bir örnek, herhangi bir zamanda bölümlerin "adil bir payının" tüketicisidir. Gruba üyeliğin sürdürülmesi süreci, Kafka protokolü tarafından dinamik olarak ele alınmaktadır. Yeni örnekler gruba katılırsa, grubun diğer üyelerinden bazı bölümleri devralacaklar; Bir örnek ölürse, bölümleri kalan örneklere dağıtılacaktır.

Kafka, bir konudaki farklı bölümler arasında değil, bir bölüm içindeki kayıtlar için toplam sipariş verir. Bölüm başına sipariş verme, verilerin anahtarla bölümlenebilme özelliği ile birlikte çoğu uygulama için yeterlidir. Ancak, kayıtlar için toplam siparişe ihtiyaç duyuyorsanız, bu yalnızca bir bölümü olan bir konu ile gerçekleştirilebilir, ancak bu tüketici grubu başına yalnızca bir tüketici süreci anlamına gelecektir.

4.3 Topic ve Kayıtlar

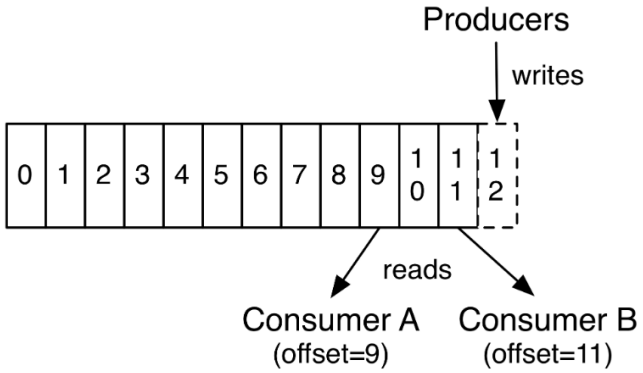
Belirli bir kategoriye ait olan mesaj akışına konu denir. Veriler topicler de saklanır.
Topicler bölümlere ayrılmıştır. Kafka her konu için bir bölümden küçük bir anahtar tutar. Bu tür bir bölüm değişmez bir sıralı dizide mesajlar içerir. Bir bölüm, eşit boyutta bir dizi segment dosyası olarak uygulanır. Kafka'daki konular her zaman çoklu abone olabilir; yani, bir konu topic, kendisine yazılan verilere abone olan sıfır, bir veya birçok tüketiciye sahip olabilir [2].
Kafka Kümesi her konu için şöyle görünen Şekil 8. de görüldüğü gibi bölümlenmiş bir kütük tutar.



Şekil 8. Topic Yapısı [2]

Her bölüm, yapılandırılmış bir işlem düğümü olarak eklenmiş düzenli ve değişmez bir kayıt dizisidir. Şekil 8’de belirtilen bölümlerdeki kayıtların her birine bölüm içindeki her kaydı benzersiz bir şekilde tanımlayan ofset adı verilen bir sıra numarası verilir [2].

Kafka kümesi, tüketilmiş olsun olmasın, yayınlanan tüm kayıtlara yapılandırılabilir bir saklama süresi kullanarak kalıcı olarak devam eder. Örneğin, saklama politikası iki güne ayarlanmışsa, o zaman bir kayıt yayınlandıktan sonraki iki gün boyunca tüketime açıktır, daha sonra yer açmak için atılır. Kafka'nın performansı, veri boyutuna göre etkin bir şekilde sabit olduğundan, verilerin uzun süre saklanması sorun değildir [2].



Şekil 9. Kafka Ofset Yapısı [2]

Tüketici bazında tutulan tek meta veriler, söz konusu tüketicinin kütük içerisindeki yeri veya dengesidir. Bu ofset tüketici tarafından kontrol edilir: normal olarak bir tüketici kayıtlarını okuduğu sırada ofsetini doğrusal olarak ilerletir, ancak aslında pozisyon tüketici tarafından kontrol edildiğinden, kayıtları istediği sırada tüketebilir. Örneğin, bir tüketici geçmişten gelen verileri yeniden işlemek ya da en son kayıtlara geçmek ve "şimdi" den tüketmeye başlamak için eski bir ofset ayarına dönebilir.

Bu özelliklerin kombinasyonu, Kafka tüketicilerinin çok kolay olduğu anlamına gelir; kümelenme veya diğer tüketiciler üzerinde çok fazla bir etkisi olmadan gelip gidebilirler. Örneğin, herhangi bir konunun içeriğini mevcut tüketiciler tarafından tüketilenleri değiştirmeden değiştirmek için komut satırı araçlarımızı kullanabilirsiniz.

Kayıttaki bölümler birkaç amaca hizmet eder. İlk olarak, günlüğün tek bir sunucuya sığacak boyutta ötesine ölçeklendirilmesine izin verirler. Her bölümün kendisini barındıran sunuculara sığması gerekir, ancak bir konunun rastgele miktarda veri işleyebilmesi için birçok bölümü olabilir.

4.5. Brokers

Brokerlar, yayınlanan verilerin korunmasından sorumlu basit bir sistemdir. Her bir broker, konu başına sıfır veya daha fazla bölüm içerebilir. Bir konuda N bölümleri ve N sayıda aracı kurum varsa, her bir aracının bir bölümü olacağını varsayalım.

Bir konuda N bölümleri varsa ve N brokerlerinden (n + m) daha fazla varsa, ilk N broker bir bölüme sahip olacak ve bir sonraki M broker bu konu için herhangi bir bölüme sahip olmayacağını düşünün.

Bir konuda N bölümleri varsa ve N brokerlerinden (n-m) daha az varsa, her broker aralarında bir veya daha fazla bölüm paylaşımına sahip olacağını varsayalım. Bu senaryo, aracı arasındaki eşit olmayan yük dağılımı nedeniyle önerilmez.

4.6. Kafka Cluster

Kafka'nın birden fazla brokerine sahip olması Kafka kümesi olarak adlandırılmaktadır. Bir Kafka kümesi, kesinti olmadan genişletilebilir. Bu kümeler, mesaj verilerinin kalıcılığını ve çoğaltılmasını yönetmek için kullanılır.

5. KAFKA STREAM İŞLEMİ

Veri akışlarını okumak, yazmak ve depolamak yeterli değildir, amaç akışların gerçek zamanlı olarak işlenmesini sağlamaktır.

Kafka'da bir stream processor, girdi konularından sürekli veri akışları alan, bu girdi üzerinde bazı işlemler yapan ve çıktı konularına sürekli veri akışları üreten herhangi bir şeydir.

Örneğin, bir perakende satış uygulaması satışların ve gönderilerin girdi akışlarını alabilir ve bu verilerden hesaplanan bir yeniden sıralama ve fiyat ayarlamaları akışı sağlayabilir.

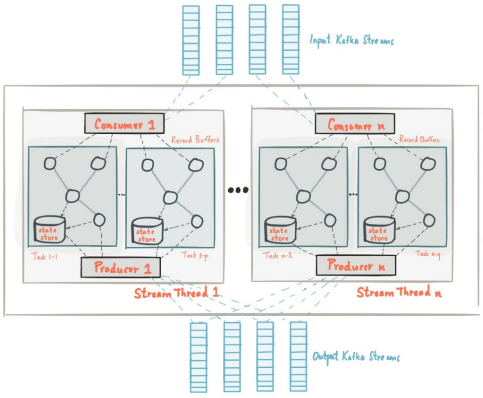
Doğrudan üretici ve tüketici API'lerini kullanarak basit işlem yapmak mümkündür. Ancak daha karmaşık dönüşümler için Kafka, tam entegre bir Streams API sağlar. Bu, akışlardan toplamaları hesaplayan veya akışlara birleştiren önemsiz işlemler yapan uygulamaların oluşturulmasına izin verir.

Bu tesis, bu tür uygulamaların karşılaştığı zor problemlerin çözülmesine yardımcı olur: sıra dışı verileri işleme, girdiyi kod değişikliği olarak yeniden işleme, durumsal hesaplamalar yapma vb.

Stream API, temel ilkellere dayanır ve Kafka'nın sağladığı içerik: üretici ve tüketici API'lerini girdi olarak kullanır, Kafka'yı durum bilgisi olan depolama için kullanır ve aynı grup mekanizmasını akış işlemci örnekleri arasındaki hata toleransı için kullanır.

5.1. Streams Architecture

Kafka Streams, Kafka üreticisi ve tüketici API'leri üzerine inşa ederek ve veri paralelliği, dağıtılmış koordinasyon, hata toleransı ve operasyonel basitlik sunmak için Kafka'nın yerel yeteneklerinden yararlanarak uygulama geliştirmeyi basitleştirir.



Şekil 10. Akış Mimarisi [2]

Şekil 10'da Kafka Streams API kullanan bir uygulamanın anatomisi gösterilmiştir. Her biri birden çok stream görevi içerir. Bu nedenle birden çok akış iş parçacığı içeren bir Kafka Stream uygulamasının mantıksal bir görünümünü sağlar [2].

5.2. Kafka Stream Api

Kafka Streams kasıtlı bir tasarımla Apache Kafka ile sıkı bir şekilde bütünleşmiştir: Kafka Streams'ın durumsal işleme özellikleri, hata toleransı ve işlem garantileri gibi birçok özelliği Kafka'nın depolama ve mesajlaşma katmanı tarafından sağlanan işlevsellik üzerine inşa edilmiştir. Bu nedenle Kafka'nın temel kavramlarını, özellikle de Kafka belgelerinde Başlarken ve Tasarım bölümlerini tanımak önemlidir.

Kafka Stream Parçaları: Kafka, üreticileri, tüketicileri ve brokerleri ayırt eder. Kısacası, üreticiler Kafka brokerlerine veri yayınlar ve tüketiciler Kafka brokerlerinden yayınlanan verileri okur. Üreticiler ve tüketiciler tamamen ayrıştırıldı ve her ikisi de Kafka kümelenmesinin çevresindeki Kafka brokerlarının dışına çıkıyor. Bir Kafka kümesi bir veya daha fazla sayıda aracı kurumdan oluşur. Kafka Streams API'sını kullanan bir uygulama hem üretici hem de tüketici olarak hareket eder.

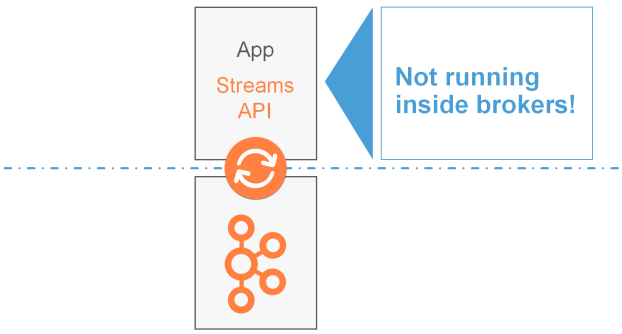
Veri: Veri topiclerde saklanır. Topic Kafka'nın sağladığı en önemli soyutlamadır: Verilerin üreticiler tarafından yayınlandığı bir kategori veya besleme adıdır. Kafka'daki her konu bir veya daha fazla bölüme ayrılmıştır. Kafka saklamak, taşımak ve çoğaltmak için verileri böler. Kafka Streams onu işlemek için veri bölümleri. Her iki durumda da, bu bölümleme esneklik, ölçeklenebilirlik, yüksek performans ve hata toleransı sağlar.

Paralellik: Kafka konularının bölümleri ve özellikle belirli bir konu için sayıları aynı zamanda Kafka'nın veri okuma ve yazma konusundaki paralelliğini belirleyen ana faktördür. Kafka ile sıkı entegrasyon nedeniyle, Kafka Akışları API'sını kullanan bir uygulamanın paralelliği temel olarak Kafka'nın paralelliğine bağlıdır[1].

5.2.1 Stream Processing Uygulaması

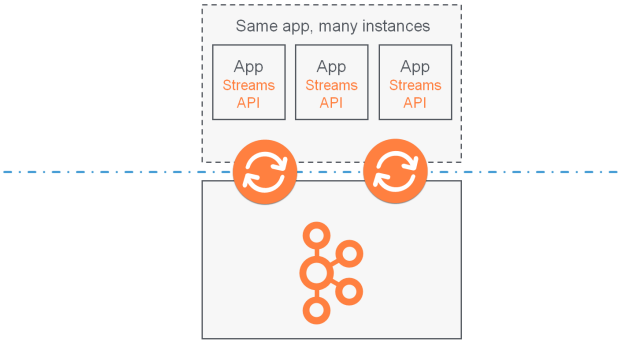
Bir Stream processing uygulaması, Kafka Streams kütüphanesinden faydalanan herhangi bir programdır. İşlemsel mantığını bir veya daha fazla işlemci topolojisi aracılığıyla tanımlayabilir.

Stream processing uygulamanız bir aracı içinde çalışmıyor. Bunun yerine, Şekil 11' de görüldüğü gibi ayrı bir JVM örneğinde veya tamamen ayrı bir kümede çalışır [1].



Şekil 11. Stream Processing Uygulaması [1]

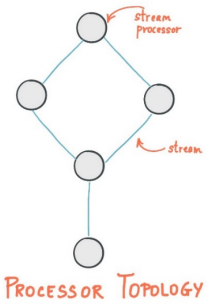
Bir uygulama örneği, uygulamanızın çalışan bir örneği veya "kopyası" dır. Uygulama örnekleri, başvurunuzu esnek bir şekilde ölçeklendirmek ve paralel hale getirmek için birincil yoldur ve aynı zamanda hataya dayanıklı hale gelmesine katkıda bulunur. Örneğin, Şekil 12'ye benzer uygulamanızın gelen veri yükünü ele almak için on makinenin gücüne ihtiyacınız olabilir; burada, uygulamanızın on örneğini, her bir makinede bir tane çalıştırmayı tercih edebilirsiniz ve bu örnekler canlı işlem sırasında yeni örnekler / makineler eklendiğinde veya mevcut olanlar çıkarılsa bile, veri işleme üzerinde otomatik olarak işbirliği yapar [1].



Şekil 12. Stream Processing Uygulaması [1]

5.2.2. Processor Topology

Bir Processor topolojisi veya basit bir topoloji, bir stream process uygulaması tarafından gerçekleştirilmesi gereken veri işlemenin hesaplama mantığını tanımlar. Şekil 12' de görüldüğü üzere topoloji, streamlere (kenarlar) birbirine bağlı stream processorların (düğümlerin) bir grafiğidir. Geliştiriciler, topolojileri düşük seviye İşlemci API'si veya öncekinin üstüne inşa eden Kafka Streams DSL ile tanımlayabilirler[2].



Şekil 13. İşlemci Topolojisi [2]

5.2.3. Stream Processor

Bir stream processor, İşlemci Topolojisi bölümündeki diyagramda gösterildiği gibi işlemci topolojisindeki bir düğümdür. Bir topolojideki bir işlem basamağını temsil eder, yani verileri dönüştürmek için kullanılır. Haritalama, birleştirme veya filtreleme gibi standart işlemler, Kafka stream processorların işlemcilerinin örnekleridir. Bir stream processor, topolojideki yukarı akış işlemcilerinden bir seferde bir giriş kaydı alır, çalışmasını ona uygular ve daha sonra aşağı akış işlemcilerine bir ya da daha fazla çıkış kaydı üretebilir.

Kafka Streams, stream processorları tanımlamak için iki API sağlar.
Birincisi, fonksiyonel DSL, çoğu kullanıcı için - ve özellikle de yeni başlayanlar için - tavsiye edilen API'dir, çünkü çoğu veri işleme kullanım durumu yalnızca birkaç DSL kod satırında ifade edilebilir. Burada, genellikle harita ve filtre gibi yerleşik işlemleri kullanırsınız.
Zorunlu, daha düşük seviyeli İşlemci API'si DSL'den daha fazla esneklik sağlar ancak daha fazla manuel kodlama çalışması gerektirmek zorunda kalır. Burada, özel işlemcileri tanımlayabilir ve bağlayabilir, ayrıca doğrudan eyalet mağazalarıyla etkileşime girebilirsiniz[2].

5.2.4. Stateful Stream Processing

Bazı stream processos uygulamaları durum durumsuzdur. Bu bir mesajın işlenmesinin diğer mesajların işlenmesinden bağımsız olduğu anlamına gelir. Örnekler, bir seferde yalnızca bir mesajı dönüştürmeniz veya bir koşulu temel alan mesajları filtrelemeniz gerektiği zamanlardır.

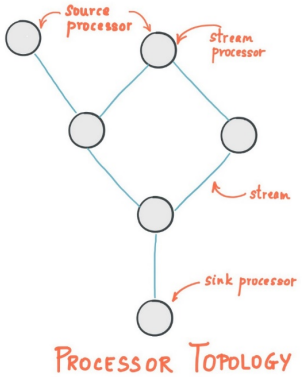
Bununla birlikte, uygulamada, çoğu uygulama durumluluk (stateful) gerektirir - doğru çalışırlar - doğru çalışabilmeleri için bu durum hataya dayanıklı bir şekilde yönetilmelidir. Uygulamanız, örneğin, girdi verilerini bir araya getirmesi, bir araya getirmesi veya pencerelemesi gerektiğinde durumsaldır. Kafka Streams, uygulamanıza güçlü, elastik, yüksek oranda ölçeklenebilir ve hataya dayanıklı durum bilgisi işleme yetenekleri sağlar[2].

5.3. Processor Topology

Bir processor topolojisi veya basit bir topoloji, uygulamanız için stream işleme hesaplama mantığını, yani girdi verilerinin çıktı verilerine nasıl dönüştürüldüğünü tanımlar. Topoloji, akışlarla (kenarlar) birbirine bağlı akış işlemcilerinin (düğümlerin) bir grafiğidir (Şekil 14). Topolojide iki özel işlemci vardır:

Source Processor: Bir source processor, herhangi bir yukarı akış işlemcisine sahip olmayan özel bir tür işlemcisidir. Şekil 14' de ki gibi bu konulardaki kayıtları kullanarak ve bunları aşağı akış işlemcilere ileterek topolojisine bir ya da daha fazla Kafka başlığından girdi akışı üretir.

Sink Processor: Sink processor, aşağı akış işlemcisi olmayan özel bir tür işlemcisidir. Şekil 14' de ki gibi yayın akış işlemcilerinden gelen tüm kayıtları belirli bir Kafka konusuna gönderir. Bir akış işleme uygulaması - yani uygulamanız - bu tür bir topolojiyi tanımlayabilir, ancak tipik olarak yalnızca bir tanesini tanımlar. Geliştiriciler, topolojileri düşük seviye İşlemci API'si veya öncekinin üstüne inşa eden Kafka Streams DSL ile tanımlayabilirler.



Şekil 14. İşlemci Topolojisi [2]

6. PARALELİZM MODELİ

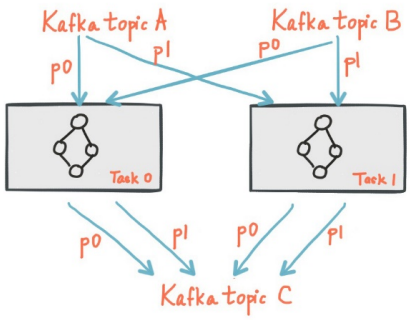
6.1. Stream Partitions and Tasks

Kafka'nın mesajlaşma katmanı saklamak ve taşımak için verileri böler. Her durumda , bu bölümleme veriye yerellik, esneklik, ölçeklenebilirlik, yüksek performans ve hata toleransı sağlayan şeydir.

Kafka Streams, akış bölümleri ve akış görevlerini paralellik modelinin mantıksal birimleri olarak kullanır. Paralellik bağlamında Kafka Streams Api ve Kafka arasında yakın bağlantılar vardır:

Her akış bölümü tamamen sıralı bir veri kayıtları dizisidir ve bir Kafka konu bölümüne eşlenir. Akıştaki veri kaydı, bu konudaki bir Kafka mesajına eşlenir.
Veri kayıtlarının anahtarları hem Kafka hem de Kafka Akımlarındaki verilerin bölümlenmesini, yani verilerin başlıklar içindeki belirli bölümlere nasıl yönlendirileceğini belirler.
Bir uygulamanın işlemci topolojisi, birden çok akış görevine bölünerek ölçeklenir. Daha spesifik olarak, Kafka Akımları, uygulama için girdi akış bölümlerine dayanan sabit bir sayıda akış görevi yaratmaktadır; bunlara, her bir göreve girdi akışlarından (yani Kafka konuları) bir bölüm listesi atanmaktadır. Akış bölümlerinin akış görevlerine atanması hiçbir zaman değişmez, bu nedenle akış görevi, uygulamanın sabit bir paralellik birimidir. Görevler daha sonra atanan bölümlere göre kendi işlemci topolojilerini başlatabilir; ayrıca, atanmış bölümlerinin her biri için bir arabellek tutarlar ve girdi verilerini bu kayıt arabelleklerinden birer birer birer birer işlerler. Sonuç olarak akış işlemleri, manuel müdahale olmadan bağımsız olarak ve paralel olarak işlenebilir.

Hafifçe sadeleştirilmiş olarak, uygulamanızın çalışabileceği maksimum paralellik, uygulamanın okuduğu giriş konusunun / bölümlerinin maksimum bölüm sayısı tarafından belirlenen maksimum akış görevi sayısıyla sınırlandırılmıştır. Örneğin, giriş konunuzda 5 bölüm varsa, o zaman en fazla 5 uygulama örneği çalıştırabilirsiniz. Bu örnekler, konunun verilerini işbirliği içinde işler. Şekil 15' de ki gibi Giriş konusunun bölümlerinden daha fazla sayıda uygulama örneği çalıştırırsanız, "fazla" uygulama örnekleri başlatılır, ancak boş kalır; ancak, yoğun örneklerden biri düşerse, boştaki örneklerden biri eskinin çalışmalarına devam eder [1].

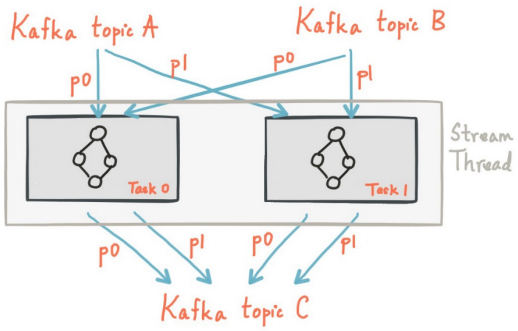


Şekil 15. Akış Bölümleri ve Görevleri [1]

Kafka Streams'ın bir kaynak yöneticisi değil, akış işleme uygulamasının çalıştığı her yerde "çalışan" bir kütüphane olduğunu anlamak önemlidir. Uygulamanın birden çok örneği aynı makinede yürütülür veya birden çok makineye yayılır ve görevler kütüphaneler tarafından otomatik olarak çalışan uygulama örneklerine dağıtılabilir. Görevlere bölümlerin atanması asla değişmez; Bir uygulama örneği başarısız olursa, atanmış tüm görevleri diğer örneklerde yeniden başlatılır ve aynı akış bölümlerinden tüketmeye devam eder.

6.2. Threading Model

Kafka Streams, kullanıcının bir uygulama örneğinde işlemeyi paralelleştirmek için kütüphanenin kullanabileceği iş parçacığı sayısını yapılandırmasını sağlar. Şekil 16' de ki gibi her iş parçacığı, işlemci topolojileri ile bağımsız olarak bir veya daha fazla akış görevi yürütebilir.



Şekil 16.Threading Modeli [3]

Daha fazla akış çekirdeği veya uygulamanın daha fazla örneğine başlamak, yalnızca topolojinin çoğaltılması ve farklı bir Kafka bölümleri alt kümesini işlemesi ve işlemeyi etkin bir şekilde paralelleştirmesi anlamına gelir. Çekirdekler arasında ortak bir durum bulunmadığına dikkat çekmek önemlidir, bu nedenle çekirdekler arası koordinasyon gerekmez. Bu, topolojileri uygulama örnekleri ve iş parçacıkları boyunca paralel olarak çalıştırmayı çok kolaylaştırır. Kafka konu bölümlerinin çeşitli akış konuları arasında atanması, Kafka'nın sunucu tarafı koordinasyon işlevselliğini güçlendiren Kafka Akımları tarafından şeffaf bir şekilde ele alınmaktadır.

Yukarıda açıkladığımız gibi, streamp process uygulamanızı Kafka Streams ile ölçeklendirmek kolaydır: yalnızca uygulamanızın ek örneklerini başlatmanız gerekir ve Kafka Akışları, uygulama örneklerinde çalışan akış görevlerinde bölüm dağıtma işini üstlenir. Uygulamanın tüm iş parçacıklarını başlatmak için Kafka konu bölümlerinin girişini kullanabilirsiniz, böylece bir uygulamanın tüm çalışan örnekleri arasında, her iş parçacığı (veya iş parçacığının yürüttüğü akış görevleri) işlenecek en az bir giriş bölümü olur[3].

7. KAYNAKÇA

[1] [Çevrimiçi]. Available: https://docs.confluent.io/current/streams/concepts.html. [Erişildi: 28 Aralık 2018].

[2] [Çevrimiçi]. Available: https://docs.confluent.io/current/streams/architecture.html. [Erişildi: 25 Aralık 2018].

[3] [Çevrimiçi]. Available: https://www.tutorialspoint.com/apache_kafka/apache_kafka_fundamentals.htm. [Erişildi: 25 Aralık 2018].