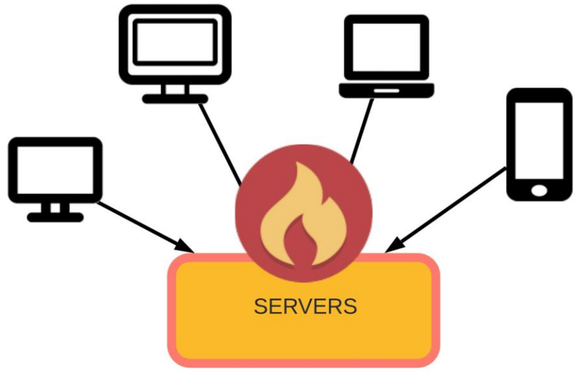
Rate Limiting

Ali Özkan

Rate Limiting Nedir?

Rate Limiting Nedir?

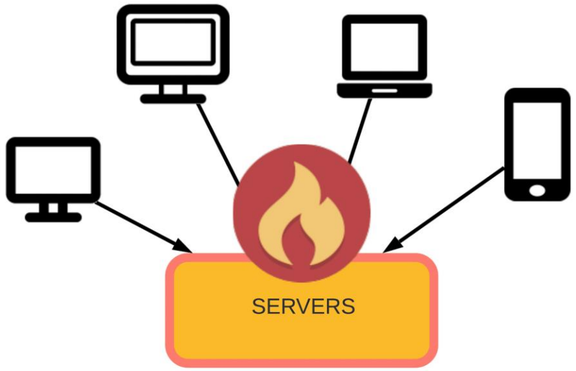
- Belirli bir zaman diliminde kabul edilen istek sayısının sınırlanması
- Sistemin aşırı yüklenmesini önleme
- Performansın ve güvenilirliğin artırılması



Rate Limiting Neden Önemlidir?

Rate Limiting Neden Önemlidir?

- DDoS saldırılarından korunma
- Bot saldırılarının engellenmesi
- API kullanımının kötüye kullanılmasının önlenmesi
- Sistem kaynaklarının verimli kullanılması



Rate Limiting Çeşitleri

Rate Limiting Çeşitleri

- Sabit Pencere (Fixed Window) Algoritması
- Kayan Pencere (Sliding Window) Algoritması
- Token Bucket Algoritması
- Leaky Bucket Algoritması
- …

Rate Limiting Çeşitleri

Sabit Pencere (Fixed Window) Algoritması

Sabit pencere algoritması, belirli bir zaman dilimindeki toplam istek sayısını takip ederek çalışan basit bir rate limiting yöntemidir.
Bu pencere, belirli bir süre boyunca sabit kalır ve bu süre zarfında izin verilen maksimum istek sayısı aşılırsa, sonraki istekler reddedilir / kuyruklanır.

Sabit Pencere Tanımlama: Belirli bir zaman dilimini (örneğin, 1 dakika) pencere olarak tanımlarız.
İstek Sayısı Sayma: Her yeni istek geldiğinde, bu isteği pencere içindeki toplam istek sayısına ekleriz.
Limit Kontrolü: Toplam istek sayısı, önceden belirlenmiş maksimum istek sayısını aşarsa, yeni gelen isteği reddederiz / kuyruğa alırız.
Pencere Tazelemesi: Belirlenen zaman diliminin sonunda, pencereyi sıfırlar ve yeni bir pencere başlatırız.

Rate Limiting Çeşitleri

Sabit Pencere (Fixed Window) Algoritması

```
// dakikada 50 istek sınırlaması
if !isRequestAllowed(request, “minute”, 50)
  return false

// saatte 1000 istek sınırlaması
if !isRequestAllowed(request, “hour”, 1000)
  return false

function isRequestAllowed(request, windowTime, maxRequestsPerWindow):
  currentWindowKey, endOfWindowSeconds = calculateWindowKey(windowTime)

  key = [apiKey | userId | IP]:[currentWindowKey] // 127.0.0.1:11:10, 127.0.0.1:11
  requestCount = Redis:incr(key)
  Redis:expire(key, endOfWindowSeconds)

  if requestCount > maxRequestsPerWindow:
    return false
  else:
    return true

function calculateWindowKey(windowTime)
  switch(windowTime)
    case “minute”
      return {currentMinute, endOfWindowSeconds} // “11:10”, 50
    case “hour”:
      return {currentHour, endOfWindowSeconds} // “11”, 3000
```

Rate Limiting Çeşitleri

Kayan Pencere (Sliding Window) Algoritması

Sabit pencere algoritmasının ani trafik artışlarına karşı daha duyarlı versiyonudur.

Pencere Boyutu Tanımlama: Belirli bir zaman dilimini (örneğin, 1 dakika) pencere olarak tanımlarız.
İstekleri Depolama: Gelen her isteği, oluştuğu zaman damgasıyla birlikte bir kuyrukta tutarız.
Limit Kontrolü: Her yeni istek geldiğinde, kuyruğun başından başlayarak pencere boyutuna eşit bir sürelik kısmını alırız (bu, mevcut penceredir). Bu penceredeki istek sayısını kontrol ederiz. Eğer maksimum istek sayısını aşarsa, yeni gelen isteği reddederiz / kuyruklarız.
Pencere Kaydırma: Yeni bir istek geldiğinde, kuyruğun başından en eski isteği çıkarırız. Böylece pencere bir adım kayar ve istediğimiz zaman dilimine odaklanırız.

Rate Limiting Çeşitleri

Kayan Pencere (Sliding Window) Algoritması

```
// dakikada 50 istek sınırlaması
if !isRequestAllowed(request, 60, 50)
  return false

// saatte 1000 istek sınırlaması
if !isRequestAllowed(request, 3600, 1000)
  return false

function isRequestAllowed(request, windowTime, maxRequestsPerWindow):

  currentTime = now.getTimestamp()
  windowStartTime = currentTime - windowTime
key = [apiKey | userId | IP]:[windowTime] // 127.0.0.1:60, 127.0.0.1:3600
  Redis:zRemRangeByScore(key, 0, windowStartTime) // pencere dışında kalan isteklerin temizlenmesi

requestCount = Redis:zCard(key) // penceredeki istek sayısı

  if requestCount > maxRequestsPerWindow:
    return false
  else:
    Redis:zAdd(key, currentTime, request.getRequestId()) // isteği pencereye ekleme
Redis:expire(key, windowTime)
    return true
```

Rate Limiting Çeşitleri

Token Bucket Algoritması

Belirli bir hızda üretilen tokenların istekler ile azaltıldığı ve token kalmadığında isteklerin engellendiği algoritmadır.

Kova Oluşturulması: Belirli bir büyüklükte bir kova oluşturulur. Bu kova, belirli bir anda tutabileceği maksimum token sayısını temsil eder.
Token Üretimi: Kova, belirli bir hızda sürekli olarak token üretir. Bu üretim hızı, sistemin izin verdiği maksimum çıkış hızını belirler.
İstek İşlemi: Bir istek geldiğinde, kovadan o isteği işleyebilmek için yeterli sayıda token alınır. Eğer yeterli token yoksa, istek reddedilir veya kuyruğa alınır.

Rate Limiting Çeşitleri

Token Bucket Algoritması

```
// maksimum 100 token biriksin, saniyede 1 token üret, her istek 1 token harcasın
if !isRequestAllowed(request, 100, 1, 1)
  return false


function isRequestAllowed(request, bucketCapacity, tokenProductionRate, requiredTokens):

  key = [apiKey | userId | IP] // 127.0.0.1
currentTime = now.getTimestamp()
lastCheckTime = Redis:get(key:lastCheckKey)

  tokensToAdd = (currentTime - lastCheckTime) * tokenProductionRate
  currentTokensInBucket = Redis:get(key)

tokensToAdd = Math.min(tokensToAdd + currentTokensInBucket, bucketCapacity - currentTokensInBucket) // hangisi küçükse

tokensInBucket = Redis:incrBy(key, tokensToAdd)

  if tokensInBucket >= requiredTokens:
    ttl = Math.toInt(bucketCapacity/tokenProductionRate)
Redis:decrBy(key, requiredTokens)
Redis:expire(key, ttl)
Redis:setEx(key:lastCheckKey, currentTime, ttl)
return true;
  else:
    return false;
```

Rate Limiting Çeşitleri

Leaky Bucket Algoritması

Bu algoritma, bir kovaya su doldurulması ve bu suyun sabit bir hızda sızdırılması mantığına dayanır. İstekler geldikçe kovaya eklenir. Kova doluysa yeni gelen istekler reddedilir. İstekler işlendikçe kovadan eksiltilir.

Kova Oluşturulması: Belirli bir büyüklükte bir kova oluşturulur. Bu kova, belirli bir anda tutabileceği maksimum istek miktarını temsil eder.
Su Doldurma: Yeni bir istek geldiğinde, kovaya eklenir. Ancak kova doluysa, bu istekler reddedilir.
Su Sızdırma: Kova, sabit bir hızda su sızdırır. Bu sızdırma hızı, sistemin izin verdiği maksimum çıkış hızını belirler.

Rate Limiting Çeşitleri

Leaky Bucket Algoritması

```
// kovada maksimum 100 birim su birikebilsin, saniyede 1 birim su sızdır, her istek 1 birim su olsun:
if !isRequestAllowed(request, 100, 1, 1)
  return false


function isRequestAllowed(request, bucketCapacity, leakRate, requiredWater):

  key = [apiKey | userId | IP] // 127.0.0.1
currentTime = now.getTimestamp()
lastCheckTime = Redis:get(key:lastCheckKey)

  waterToDrain = (currentTime - lastCheckTime) * leakRate

bucketLevel = Math.max(0, bucketCapacity - waterToDrain) // hangisi büyükse

  if bucketLevel + requiredWater <= bucketCapacity:
    ttl = Math.toInt(bucketCapacity/tokenProductionRate)
Redis:incrBy(key, requiredWater)
Redis:expire(key, ttl)
Redis:setEx(key:lastCheckKey, currentTime, ttl)
return true;
  else:
    return false;
```

Rate Limiting Çeşitleri

Geçersiz token’lar ile istek gelen ip adresi için rate limit uygulayalım ve limiti aştıktan sonra artık token kontrolü yapmadan belirli bir süre boyun

```
if isBanned(request):
   return false;
rateLimitConf = { // 10 dakikada 30 farklı geçersiz token gelirse ip adresini 1 saatliğine yasakla
   invalidTokenDuration: 600,
   maxInvalidTokens: 30,
   jailDuration: 3600,
}
if !checkToken(request)
   invalidTokenSent(request, rateLimitConf)
   return false;

function isBanned(request):
  bannedKey = banned:[request.IP] // banned:127.0.0.1

  if Redis:get(bannedKey): // zaten yasaklı
    return true
  else:
   return false;

function invalidTokenSent(request, rateLimitConf):

  tokenKey = invalidTokens:[request.IP] // invalidTokens:127.0.0.1
Redis.sAdd(tokenKey, request.token)
Redis.expire(tokenKey, request.token, rateLimitConf.invalidTokenDuration)
invalidTokenCount = Redis:sCard(tokenKey)

  if invalidTokenCount >= rateLimitConf.maxInvalidTokens:
    bannedKey = banned:[IP] // banned:127.0.0.1
    Redis:set(bannedKey, currentTime, [ex: rateLimitConf.jailDuration])
```

Dikkat Edilmesi Gerekenler

- CGN IP Havuzu

ISP’ler farklı lokasyonlardaki kullanıcıları tek bir ip adresinden çıkartıyor olabilir.

Dikkat Edilmesi Gerekenler

- IP ByPassing

```
curl -X GET --location 'https://api.example.com/v1/content/1' \
--header 'X-Originating-IP:  127.0.0.1' \
--header 'X-Forwarded-For:  127.0.0.2' \
--header 'X-Remote-IP:  127.0.0.3' \
--header 'X-Remote-Addr:  127.0.0.4' \
--header 'X-Client-IP:  127.0.0.5' \
--header 'X-Host:  127.0.0.6' \
--header 'X-Forwared-Host:  127.0.0.7' \
--header 'X-Forwarded-For:  127.0.0.8’
```

Dikkat Edilmesi Gerekenler

- Auto-Scaling

Rate limit sayaçlarının yerel kullanılması, birden fazla sunucu durumunda Rate Limit'in olması gerektiği gibi çalışmaması.
Redis, memcache, SQL gibi merkezi depolama alanları kullanımı.

- Race Condition
Önce set et / arttır, sonra al

Dikkat Edilmesi Gerekenler

- Endpoint’in query/post parametreleri ile rate limit anahtarı olarak kullanılması

POST /forgot-password?fake=1 HTTP/1.1

Host: target.com

email=victim@gmail.com&alsofake=2

Rate Limiting

Teşekkürler

in: aliozkan

x: aliozkan_