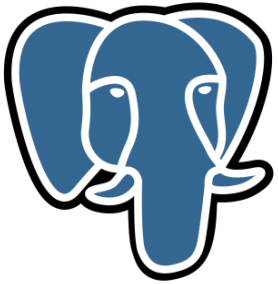


Performans Optimizasyonu ve Büyük Ölçekli Sistemler

Quality of Service in PostgreSQL

Şeyma Mintaş

Cooksoft Teknoloji AŞ

seyma.mintas@cooksoft.com.tr


seyma-mintas

Sunum İçeriği

Performans Nedir?

Performans Nedir?

Latency

Bir isteğin başlangıçtan bitişe kadar geçen süre.

Genel kullanımda ms cinsinden ölçülür. Kullanıcı 'hız' olarak bunu hisseder.

Throughput

Birim zamanda işlenen iş miktarı.

TPS (transaction/sn), QPS (sorgu/sn). Sistem kapasitesini gösterir.

Resource Utilization

CPU, bellek, disk I/O, ağ.

Bir kaynak %100 dolduğunda diğer her şey yavaşlar — darboğaz yaşanır.

Neden Zordur?

Performans optimizasyonu tek boyutlu değildir. Latency'yi düşürürken throughput'u artırmak, kaynakları verimli kullanmak ve bunu tutarlı yapmak — hepsi aynı anda hedeflenmelidir.

Veritabanında Performans

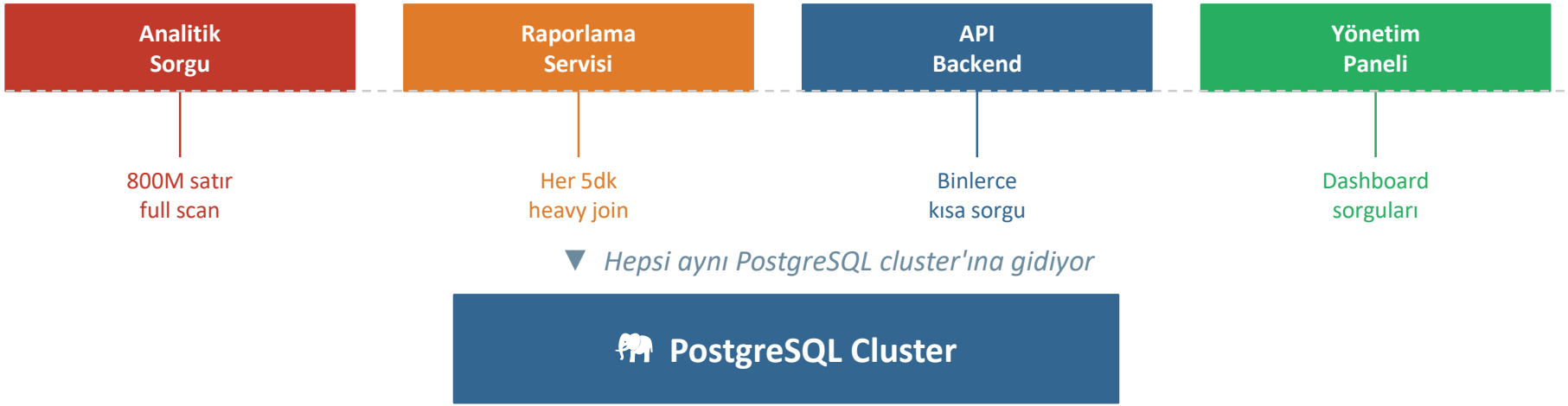
PostgreSQL perspektifinden

Veritabanında Performans Katmanları

Sorgu Katmanı	Planner, index kullanımı, join stratejisi, istatistikler
Bellek Katmanı	shared_buffers, work_mem, wal_buffers, effective_cache_size
I/O Katmanı	Disk erişimi, WAL yazımı, checkpoint, random_page_cost
Bağlantı Katmanı	Her bağlantı = process. Connection overhead, pooling ihtiyacı
Eşzamanlılık	MVCC, lock contention, dead tuple birikimi, VACUUM

PostgreSQL'de Performans: Gerçek Sorun

Tek Cluster — Çok Kullanıcı



Sonuç: Eşitsiz Rekabet

Analitik sorgu tüm CPU ve memory'yi kullanıyor → API backend yavaşlıyor → kullanıcı deneyimi bozuluyor.
Hiç kimse birbirinin farkında değil.

Öncelik & QoS Kavramı

Quality of Service nedir, neden gerekir?

Quality of Service

Quality of Service (QoS)

Bir sistem içinde farklı iş yükleri ve kullanıcılar arasında kaynakları kontrollü, adil ve öngörülebilir şekilde dağıtma mekanizmasıdır.

Gerçek Dünyadan Örnekler:

Uçak İçi Koltuk Sınıfları	Business class → önce iner, daha geniş alan, garanti hizmet. Economy class → aynı uçak, farklı SLA.
İnternet Servis Sağlayıcı	VoIP trafiği videoya göre önceliklidir. Paket kaybı tolere edilemez, gecikme tolere edilemez.
Acil Servis Triaj	Kalp krizi → anında müdahale. Soğuk algınlığı → sıraya gir. Kaynaklar sınırlı, öncelik hayat kurtarır.

Öncelik Türleri: Ne Kontrol Ederiz?

CPU Önceliği

İşlemci zamanının kime ne kadar verileceği.
OS scheduler seviyesinde: nice value, cgroup, CPU affinity.

Bellek Önceliği

work_mem, shared_buffers — hangisi ne kadar ayırabilir?
Limit aşılınca disk'e döküm (spill) → ağır gecikme.

Eşzamanlılık Limiti

Kaç sorgu aynı anda çalışabilir?
Sınırsız eşzamanlılık = lock contention = herkes yavaşlar.

I/O Önceliği

Disk okuma/yazma bant genişliği.
Bir backup tüm I/O'yu doldurursa OLTP etkilenir.

Bağlantı Kotası

Bir rol ya da veritabanı kaç bağlantı açabilir?
Sınırsız = connection exhaustion = sistem çöküşü.

Zaman Limiti

Bir sorgu ne kadar süre çalışabilir?
Runaway query'leri durdur, kaynak serbest bırak.

Önceliği Neye Göre Belirliyoruz?

Önceliklendirme Mekanizmaları

İşletim Sistemi Seviyesi

nice / renice

Process CPU önceliği (-20 en yüksek, +19 en düşük). PostgreSQL backend'leri farklı öncelikle çalıştırılabilir.

CPU Affinity

Bir process hangi CPU çekirdeklerinde çalışabilir? sched_setaffinity() ile backend'i belirli core'lara bağla.

ionice

Disk I/O önceliği. Backup process'ini düşük önceliğe alarak OLTP'yi koru.

Veritabanı Seviyesi

statement_timeout

Sorgulara maksimum çalışma süresi. Runaway query'leri öldür.

lock_timeout

Lock bekleme süresi limiti. Deadlock riskini azalt.

idle_in_transaction_timeout

Boşta kalan transaction'ları temizle.

work_mem

Her sort/hash işlemi için bellek. Rol bazlı ayarlanabilir.

connection limits

CREATE ROLE ... CONNECTION LIMIT N ile bağlantı kotası.

max_parallel_workers

Paralel sorgu worker sayısı. Ağır sorgular için CPU kullanımını sınırla.

Sınırlar ve Sorunlar

Bu yaklaşımlar tek başına yeterli mi?

OS Araçları Kör

nice/cgroups PostgreSQL'in içini görmez. Hangi sorgu kritik, hangisi değil? OS bilmiyor. Granülite yok.

GUC'lar Statik

statement_timeout tüm sorgulara eşit uygulanır. Role bazlı kısmi destek var ama dinamik değil, izleme yok.

Eşzamanlılık Kontrolü Eksik

PostgreSQL'de yerleşik olarak 'şu anda kaç SELECT çalışıyor, sınır koy' mekanizması yoktur.

Görünürlük Yok

Hangi rol hangi kaynağı ne kadar kullanıyor? Standart araçlarla gerçek zamanlı kaynak kullanımı izlenemez.

Sonuç

Gerçek anlamda önceliklendirme için veritabanı motoruyla entegre, sorgu düzeyinde farkındalığa sahip bir kaynak yöneticisi gerekir.

PostgreSQL Ekosisteminde QoS

Mevcut Çözümler

1 PostgreSQL Yerleşik Mekanizmalar

```
-- Role bazlı GUC ayarları:
ALTER ROLE analytics_user SET statement_timeout = '30s';
ALTER ROLE analytics_user SET work_mem          = '64MB';
ALTER ROLE analytics_user SET lock_timeout      = '5s';

-- Bağlantı limiti:
ALTER ROLE reporting_user CONNECTION LIMIT 10;
```

✓ Artılar

- Role bazlı GUC: basit, yerleşik
- CONNECTION LIMIT: bağlantı kotası
- Restart gerekmez

✗ Eksikler

- CPU, eşzamanlılık kontrolü yok
- Dinamik limit değişikliği zor
- Monitoring entegrasyonu eksik

Mevcut Çözümler: PgBouncer & Resource Groups

2 PgBouncer — Bağlantı Seviyesi Kontrol

```
# pgbouncer.ini — pool bazlı limitler:
[databases]
analytics_db = host=localhost dbname=mydb pool_size=5
api_db       = host=localhost dbname=mydb pool_size=50

# Pool mode seçimi:
pool_mode = transaction    # En verimli
```

PgBouncer ne yapar?
- Bağlantı havuzlama → process overhead ortadan kalkar
- Pool size = eşzamanlı bağlantı sınırı
- Farklı database → farklı pool boyutu

Ne yapmaz?
- CPU/bellek limiti yok
- Sorgu düzeyinde farkındalık yok

3 Citus / Patroni Ekosistemindeki Yaklaşımlar

Citus Worker Limits

Multi-node PostgreSQL. Sorguları worker'lara dağıtır. Kaynak kotası sınırlı.

pg_query_settings

Sorgu pattern bazlı GUC override. Statik, query fingerprint eşleşmesi.

Patroni / pgBadger

HA + log analizi. Proaktif kaynak yönetimi değil, reaktif.

Mevcut Çözümler: Karşılaştırma Tablosu

Araç	CPU	Bellek	Eşzamanlılık	Bağlantı	Sorgu-Aware	Dinamik
PostgreSQL GUC'lar	✗	⚠	✗	⚠	⚠	⚠
PgBouncer	✗	✗	⚠	✓	✗	⚠
cgroups (OS)	⚠	⚠	✗	✗	✗	⚠
pg_query_settings	✗	⚠	✗	✗	⚠	✗
pg_qos	✓	✓	✓	✓	✓	✓

Yukarıdaki araçların hiçbiri tek başına: hem CPU hem bellek hem eşzamanlılık hem de bağlantı kotasını, sorgu düzeyinde farkındalıkla, dinamik olarak yönetemiyor. İşte pg_qos tam bu boşluğu dolduruyor.

pg_qos: Deep Dive

github.com/appstonia/pg_qos

pg_qos: Nedir, Ne Yapar?

pg_qos

AppstoniA tarafından geliştirilen, PostgreSQL 15+ için Quality of Service kaynak yönetim eklentisi.
Session, sorgu ve transaction bazlı CPU, bellek ve eşzamanlılık limitleri sağlar. Rol ve veritabanı bazlı izolasyon sunar. Reconnect gerektirmez.

CPU Affinity	work_mem Limiti	Eşzamanlılık	Hızlı Propagasyon
Linux sched_setaffinity Backend → N çekirdeğe bağlanır Parallel worker'lar da kapsanır	SET work_mem intercept edilir Limit üstü değer reddedilir Bellek izolasyonu garantili	SELECT / INSERT / UPDATE DELETE / TX (transaction) bazında limit aşımında hata ver	Shared epoch mekanizması Limit değişikliği anlık yayılır Reconnect gerekmez!

Gereksinimler: PostgreSQL 15+ • Linux (CPU için) • shared_preload_libraries

pg_qos: Kurulum

1 Kurulum

```
# Ubuntu 24.04:
sudo dpkg -i postgresql-17-qos_1.0.0-1-
ubuntu24_amd64.deb
sudo apt-get install -f

# RHEL / AlmaLinux 10:
sudo rpm -i postgresql17-qos-1.0.0-1.el10.x86_64.rpm

# Kaynak koddan:
git clone https://github.com/appstonia/pg_qos
make && sudo make install
```

2 Aktifleştirme

```
# postgresql.conf:
shared_preload_libraries = 'qos'
# → Server restart gerektirir!

# Her veritabanında:
CREATE EXTENSION qos;

# Debug modu:
SET client_min_messages = 'debug1';
```

Desteklenen Platformlar

Debian 13 • Ubuntu 24.04 • RHEL/AlmaLinux/CentOS 10 (PostgreSQL 15 ve üzeri kurulup çalıştırılabilen tüm Linux versiyonlarında kaynak koddan kurulum yapılıp çalıştırılabilir.

PostgreSQL 15, 16, 17, 18 • CPU limiting yalnızca Linux'ta (diğer platformlarda parallel worker limiti uygulanır)

pg_qos: Parametreler & Kullanım

Parametre	Tip	Ne Sınırlar	Örnek
qos.cpu_core_limit	integer	Max CPU çekirdek (affinity)	2
qos.work_mem_limit	bytes	Max work_mem / session	64MB
qos.max_concurrent_tx	integer	Max eşzamanlı transaction	50
qos.max_concurrent_select	integer	Max eşzamanlı SELECT	40
qos.max_concurrent_update	integer	Max eşzamanlı UPDATE	10
qos.max_concurrent_insert	integer	Max eşzamanlı INSERT	20
qos.max_concurrent_delete	integer	Max eşzamanlı DELETE	5

pg_qos: Parametreler & Kullanım

```
-- Role bazlı limit:
ALTER ROLE analytics SET qos.cpu_core_limit = '2';
ALTER ROLE analytics SET qos.work_mem_limit = '128MB';
ALTER ROLE analytics SET qos.max_concurrent_select = '5';

-- DB bazlı limit:
ALTER DATABASE tenant_42 SET qos.max_concurrent_tx = '20';

-- Role+DB kombinasyonu (en kısıtlayıcı uygulanır):
ALTER ROLE app IN DATABASE tenant_42 SET qos.work_mem_limit = '4MB';


test=# select * from qos_settings;
    rolname    | datname |           setting
---------------+---------+-----------------------------
 database-wide | test    | qos.cpu_core_limit=2
 database-wide | test    | qos.max_concurrent_tx=4
 database-wide | test    | qos.max_concurrent_update=2
 database-wide | test    | qos.max_concurrent_insert=2
 database-wide | test    | qos.max_concurrent_delete=1
```

pg_qos Roadmap

QoS nereye gidiyor?

pg_qos Roadmap: Gelecekte Neler Geliyor?

Limit Duration

Yakın Dönem

Şu anda eş zamanlı limitleme yapılıyor. Limiti belirli bir zaman aralığı içerisinde gerçekleştirme.
Örneğin: 2 sn içerisinde 5 select sorgusu çalıştırılabilir.
10 sn içerisinde 10 transaction çalıştırılabilir.

Ağırlıklı Önceliklendirme

Orta Dönem

CPU Affinity yerine oransal olarak CPU kaynak limitlemesi yapılabilir.(%10-%20)

Gerçek Zamanlı Monitoring View

Yakın Dönem

pg_qos_stats view: Her rol ve DB için anlık kaynak kullanımı, limit aşım sayısı, bekleme süresi. Prometheus exporter ile entegrasyon.

Limit Burst

Uzun Dönem

Belirlenmiş limit aşımında hata döndürmek yerine kuyruğa alınabilir.

Daha Büyük Resim: PostgreSQL'de QoS Nereye Gidiyor?

Topluluk & Standartlaşma

PostgreSQL Core'a Entegrasyon

pg_qos gibi extension'ların kanıtladığı ihtiyaç, ileride core'a taşınabilir. MySQL'in Resource Groups özelliği bu yolda. PostgreSQL topluluğu tartışıyor.

Kubernetes & Cloud Native

Containerized PostgreSQL'de QoS kritik. pg_qos + Kubernetes resource requests/limits = çift katman isolation. PgBouncer + pg_qos = tam yığın.

Multi-Tenant SaaS Standartı

Her tenant'a ayrı instance yerine tek cluster + pg_qos. Bakım kolaylığı, maliyet düşüklüğü ve SLA garantisi bir arada. Endüstri trendi bu yönde.

Bugün bir extension olarak sunulan bu yetenek, yarın PostgreSQL'in vazgeçilmez bir parçası haline gelebilir. Şu an bunu erken adopter olarak kullanıyorsunuz.

Teşekkürler!

Sorular?

pg_qos: github.com/appstonia/pg_qos | Şeyma Mintaş · Cooksoft | konferans.postgresql.org.tr