Avrupa Bilim ve Teknoloji Dergisi
Özel Sayı, S. 310-317, Nisan 2020

Araştırma Makalesi

www.ejosat.com ISSN:2148-2683

European Journal of Science and Technology
Special Issue, pp. 310-317, April 2020

Research Article

ORCA: GO Programlama Dili için ORM/ODM Kütüphanesi**

Mehmet Emin Kaymaz[1], Övünç Öztürk[2*]

[1] Manisa Celal Bayar Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Manisa, Türkiye (ORCID: 0000-0002-0710-5732)
[2] Manisa Celal Bayar Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Manisa, Türkiye (ORCID: 0000-0001-7127-7902)

(Konferans Tarihi: 5-7 Mart 2020)

(**DOI:** 10.31590/ejosat.araconf40)

ATIF/REFERENCE: Kaymaz, M. E. & Öztürk, Ö. (2020). ORCA: GO Programlama Dili için ORM/ODM Kütüphanesi. *Avrupa Bilim ve Teknoloji Dergisi*, (Özel Sayı), 310-317.

Öz

Bu bildiride Go olarak bilinen Golang dili için geliştirilmiş ORCA adında yeni bir ORM kütüphanesi sunulmaktadır. Geliştirilen kütüphane SQLite, MongoDB, RedisDB ve MemcacheDB olmak üzere dört farklı veritabanı sistemine veritabanı tipinden bağımsız olarak destek sunmaktadır. Böylelikle rakiplerinden farklı olarak, veritabanı ve hatta veritabanı tipi değiştirildiğinde bile uygulamanın kaynak kodunu değiştirmeden kullanabilme imkanı sunmaktadır. Bunun dışında benzerlerine göre daha kullanıcı dostu ve daha yüksek performans sağlayan bir kütüphanedir. Ayrıca kullanıcının kendi fonksiyonlarını yazabilmesini sağlayan yerel kanca (local hook) ve global kanca (global hook) yapılarını sunmaktadır.

Anahtar Kelimeler: ORM Kütüphanesi, Go, Golang, veritabanı, GORM, XORM.

ORCA: An ORM/ODM Library for Go Programming Language

Abstract

This paper proposes a new ORM library, namely ORCA, for the Golang (Go) language. The proposed library supports four different database systems (SQLite, MongoDB, RedisDB and MemcacheDB), regardless of database type. Thus, unlike its competitors, it offers the possibility to use the database without changing the source code ıf when the database or even the database type is changed. Besides, ORCA performs better with a more user friendly syntax compared to its competitors. ORCA also offers local hook and global hook structures that enable the user to write their own functions.

Keywords: ORM Library, Go, Golang, database, GORM, XORM.

* Sorumlu Yazar: Manisa Celal Bayar Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Manisa, Türkiye, ORCID: 0000-0001-7127-7902, ovunc.ozturk@cbu.edu.tr

** Bu bildiri ***International Conference on Access to Recent Advances in Engineering and Digitalization (ARACONF 2020)*** de sunulmuştur.

1. Giriş

Go (Golang) Google tarafından geliştirilen, statik olarak derlenmiş bir açık kaynak programlama dilidir. Eş zamanlı programlamayı destekler diğer bir deyişle kanalları, "goroutines"leri vb. Bileşenleri kullanarak eş zamanlı olarak birçok işlemi çalıştırabilir. Go bellek yönetimini kendisi sağlayan ve ertelenmiş fonksiyonları destekleyen bir çöp toplama sistemine sahiptir.

Go programlama dili temelde Google'ın arka plandaki sistem performansını arttırmak amacıyla geliştirilmiş bir sistem programlama dilidir. Fakat az kaynak tüketmesi, web uygulamalarının kendi kendini host ediyor olması ve yüksek performans özellikleri sayesinde Web programlama tarafında yoğun ilgi görmektedir. Go dilinin özelliklerini sıralamak gerekirse, Diğer pogramlama dillerinin aksine Go dilinde koleksiyon yapıları bulunmamaktadır. "Net", "os" gibi güçlü kütüphanelere sahiptir. Diğer diller ile sistem çağrısı komutları kullanarak yapılan bir kısım iş Go ile platform bağımsız çalışabilecek şekilde yazılabilir. Sistem çağrısı komutu işletim sistemi ile doğrudan iletişim kurmayı ve işletim sistemine iş yaptırmayı sağlar. Ancak her işletim sisteminin farklı arayüzlere sahip olduğundan sistem çağrısı komutu kullanılarak yazılan bir program sisteme bağımlı kalmaktadır. Go diline ait "net" ve "os" gibi kütüphaneler ile bu komutların bir kısmını işletim sisteminden bağımsız olarak kullanılabilmektedir.

[1], Web uygulamalarının karşılaştırılması için sunulmuş açık kaynaklı Acme Air uygulamasını [2], [3] kullanarak Go, Javascript ve Java programlama dillerinin performanslarını karşılaştırmaktadır. Go programlama dilinin diğer ikisine göre önemli ölçüde iyi performans gösterdiği görülmüştür. Deneylerde, Java'nın Web çerçevesinde Go'dan 10 kat daha fazla zaman harcadığı gözlemlenmiştir. Deneysel sonuçlar, Go gibi statik olarak derlenmiş dillerin performansının cazip olduğunu göstermiştir.

Bunun yanısıra Go dili mikroservis tabanlı mimariyi desteklediği ve Docker gibi konteyner uygulamalarının altyapısında kullanıldığı için Web uygulamasının tümünü Go diliyle geliştirmek yerine JavScript vb. dillerle hibrit olarak uygulama geliştirmeyi de destekler. Böylelikle diğer dillerde geliştirilmiş uygulamaların yoğun zaman alan kısımlarını Go diliyle yeniden yazarak, az emekle önemli performans kazancı elde edilebilir.

Yukarıda bahsedilen nedenlerle Go programlama dilinin Web programlana dilleri arasında popülerliği gittikçe artmaktadır. 2017'nin başlarında TIOBE gibi dil popülerlik listeleri Go dilini en popüler 14. dil olarak belirlemiştir [4].

Nesne ilişkisel eşleyici (ORM), ilişkisel veritabanları tablolarında depolanan verilerin uygulama kodunda daha sık kullanılan nesnelere aktarılmasını otomatikleştiren bir kod kütüphanesidir. Bu çalışmada, Go programlama dili için önerilmiş yeni bir ORM kütüphanesi sunulmaktadır. Önerilen ORM kütüphanesinin mevcut kütüphanelere göre üstün yönleri daha kullanıcı dostu olması, daha yüksek performans sunması, kullanıcının kendi fonksiyonlarını yazabilmesi ve aynı kodu farklı veritabanlarında ve veritabanı tiplerinde kullanılmasını desteklemesi olarak sıralanabilir.

Bir sonraki bölümde, önerilen kütüphanenin mimarisi ve öne çıkan özellikleri listelenmektedir. Daha sonraki bölümde, önerilen kütüphanenin başarımı test edilmiş ve rakipleri ile karşılaştırılmıştır. Son bölüm, sonuçları ve olası gelecek çalışmaları listelemektedir.

2. Materyal ve Metot

2.1. GO Dili İçin Diğer ORM Kütüphaneleri

Literatürde yaygın olarak kullanılan açık kaynak iki ORM kütüphanesi bulunmaktadır: GORM [5] ve XORM [6]. GORM kütüphanesi PostgreSQL, SQLite, MySQL, MSSQL veritabanlarını desteklerken, XORM kütüphanesi ise TiDB, ORACLE, MySQL, MSSQL veritabanlarını desteklemektedir. Go programlama dilinde en kapsamlı veritabanı desteği bulunan ORM araçları olarak GORM ve XORM kütüphaneleri örnek verilebilirken, daha kısıtlı veritabanı desteği bulunan çeşitli ORM ve ODM araçları da literatürde mevcuttur. Örneğin, Kallax [7], Redis-ORM [8], Mongodm [9] ve pg[13] bunlara örnek verilebilir. Go programlama dili için geliştirilmiş ORM kütüphanelerine yakından bakacak olursak,

Tablo 1. Literatürdeki ORM Kütüphaneleri

ORM Kütüphaneleri	Veritabanı sayısı	Veritabanı tipi sayısı	Veritabanı isimleri
ORCA [10]	4	3	SQLite, MongoDB, RedisDB, MemcacheDB
GORM [5]	4	1	SQLite, MySQL, PostgreSQL, MSSQL
XORM [6]	4	1	TiDB, ORACLE, MySQL, MSSQL
Kallax [7]	1	1	PostgreSQL
Pg [11]	1	1	PostgreSQL
RedisORM [8]	1	1	RedisDB
MongoDM [9]	1	1	MongoDB

Tablo-I'de görüldüğü üzere şuan en fazla veritabanı desteği veren ORM kütüphanelerinin GORM,XORM ve ORCA olduğu ve diğer kütüphanelerin ise belli bir veritabanına yönelik destek sunduğu görülmüştür.

En geniş veritabanı desteği veren ORM kütüphanelerine yakından bakarsak her birinin eşit sayıda veritabanı desteği verdiği ancak ORCA'nın 3 farklı veritabanı tipini desteklediği görülmüştür ve bu durum kullanıcıya büyük bir avantaj sağlamaktadır.Örneğin, kullanıcı eğer isterse programın kaynak kodunu değiştirmeden SQL veritabanından NoSQL veritabanına ORCA sayesinde kolayca geçiş sağlayabilir.

Ayrıca ORCA'nın spesifik tek bir veritabanına destek veren bir ORM kütüphanesiyle karşılaştırıldığında da kullanıcıya geliştirme kolaylığı ve avantajlar sağladığı görülür.Örneğin, ORCA Redis veritabanı üzerinde ilişkisel olarak anahtar isimlendirmesini kullanıcıya sağlayabilirken, Kallax gibi sadece RedisDB desteği veren bir ORM kütüphanesinde bile böyle bir özellik bulunmamaktadır.

Go programlama dilini kullanan geliştiriciler için en fazla veritabanı desteği sunan ve farklı tipte veritabanlarını destekleyen ORM kütüphanesi ORCA'dır.

Tablo-II'de görüldüğü gibi ORCA, GORM ve XORM kütüphanelerinin her biri 4 veritabanına destek vermekte ancak ORCA farklı veritabanı tiplerine de destek vererek veritabanı tarafında yapılacak ciddi bir değişimde bile kaynak kodda yapılacak değişikliği minimize ederek tekrar kullanılabilirliğini sağlamaktadır.

Tablo 2. Literatürdeki ORM Kütüphaneleri

Özellikler	**ORCA**	**GORM**	**XORM**
Desteklenen Veritabanı sayısı	4	4	4
SQL Veritabanı desteği	+	+	+
NoSQL-Document Veritabanı desteği	+	-	-
NoSQL-Key-Value Veritabanı desteği	+	-	-

2.2. ORCA ORM Kütüphanesi

ORCA güncel sürümünde dört farklı veritabanı desteği sunmaktadır. Bu veritabanları SQLite (SQL), MongoDB(NoSQL-Document), RedisDB (NoSQL-KeyValue Store) ve Memcached (NoSQL-KeyValue Store) olarak sayılabilir. ORCA'nın diğer kütüphanelerden ayrıldığı temel nokta hem ORM hem de ODM kütüphanesi olarak davranabilmesi ve bunu kodda minimum değişiklikle gerçekleştirmesidir. Bu nedenle alt seviyede kullanılan veritabanının değişmesi durumunda kodda değişiklik yapılmamasını sağlıyor. Bu nedenle Polyglot persistence'ın [12] uygulanmasını kullanılması gereken kütüphane sayısını azaltarak kolaylaştırmaktadır.

Bunun yanında ORCA'nın diğer bir katkısı da veritabanı işlemlerini daha kullanıcı dostu bir sözdizimi ve komut yapısı ile gerçekleştirmeyi mümkün kılmasıdır. ORCA ile fonksiyonel dillerde temel yapı taşı olan ve tüm dillerde kullanılan liste tipi kullanarak tüm veritabanı işlemlerinin gerçekleştirilmesi sağlanmaktadır. Örneğin, temel CRUD işlemlerini gerçekleştirmek toplam 25 satır uzunluğunda bir kod ile gerçekleştirilebilir. Geliştirilen kütüphanenin temel amacı Go dili ile farklı veritabanlarının daha kolay ve verimli bir şekilde kullanılmasını sağlayacak bir programlama arayüzü sunmaktır.

2.3. Temel Nitelikler

2.3.1. Auto Migrations

ORCA kullanıcının tanımladığı verinin özelliklerine bakarak bu verinin veritabanı tarafında varsa model ilişkileri kurularak saklanmasını sağlar böylece kullanıcının tablo veya koleksiyon (veritabanına göre değişecektir) oluşturması ya da model ilişkilerini belirtmesine gerek kalmaz.

Kullanıcı sadece kullanmak istediği veri tipini Go dilinde tanımlar ve ORCA'yı kullanarak veritabanı işlemlerini gerçekleştirir ve bu işlemler gerçekleşirken ORCA kullanmakta olduğu veri tipini analiz edip verinin veritabanı üzerine modellenmesini sağlar.Böylece kullanıcı yazmakta olduğu algoritmaya odaklanabilir ve verinin nasıl ve ne şekilde veritabanında saklanıp modelleneceğiyle ilgilenmesine gerek kalmaz.

Eğer veritabanında ORCA tarafından modellenmiş ve saklanmış veriler varsa ORCA bu verileri ilgili veritabanından okuyarak belleğine (Cache) alacaktır ve bu okunan veriler kullanıcının tanımlamış olduğu veri tipleriyle otomatik olarak ilişkilendirilecektir.

Bu anlatılan işlemlerin tümü kullanıcı tarafından ORCA kullanılarak tek bir kod satırıyla gerçekleştirilebilir.

2.3.2. Composite Primary Key

ORCA kullanıcının tanımlamış olduğu verinin ID değerini otomatik olarak oluşturur.Böylece kullanıcı herhangi bir ID değeri belirtmek zorunda kalmadan verinin kendisini vererek işlemini gerçekleştirebilir.Örneğin, kullanıcı kendi tanımlamış olduğu veri tipinden bir değişken oluşturmuş olsun ve ilgili değişken ORCA kullanılarak veritabanına kaydedilsin bu durumda kullanıcı kaydetmiş olduğu bu veriyi eğer güncellemek ya da silmek isterse tanımlamış olduğu değişkeni ORCA'ya vererek veritabanı işlemini gerçekleştirebilir.

Bunun yanında ORCA SQL veritabanlarında yabancı anahtar (Foreign Key) tanımlama ve ilişkilendirme işlemlerini kullanıcının tanımladığı veri tipine göre arka planda gerçekleştirir.

2.3.3. Composite List

Günümüzde kullanılan çoğu programlama dili koleksiyonları desteklemekte haliyle birçok geliştirici bu koleksiyonların nasıl kullanılacağını bilmekte bu yüzden ORCA kullanılması ve öğrenilmesi kolay olması amacı ile bir koleksiyon gibi tanımlanıp kullanılmasına imkan verecek bir yapıda geliştirilmiştir ve bu yapı Composite List'dir.

Bileşik liste (Composite List) veritabanındaki bir tablo ya da bir koleksiyonu temsil eder.Bileşik listenin elemanları, temsil ettiği tablo ya da koleksiyonda bulunan verilerdir.Kullanıcı bileşik listenin elemanları üzerinde bir değişiklik yaptığında bu değişiklik anında veritabanına iletilir.Kullanıcıya bileşik liste elemanları üzerinde değişiklik yapabilmesi için sağlanan bazı metotlar vardır.

Bu metotlar,

- **AddRange**, veritabanına tek bir transaction altında birden fazla kayıt eklemek için kullanılır.
- **Add**, veritabanına tek bir transaction altında bir kayıt eklemek için kullanılır.
- **Update**, bir transaction altında veritabanındaki bir veriyi güncellemek için kullanılır.
- **Delete**, veritabanından bir veriyi silmek için kullanılır.
- **Clear**, bileşik listenin temsil ettiği tablo ya da koleksiyonun tüm içeriğini silmek için kullanılır.
- **Foreach**, kullanıcının tanımladığı bir fonksiyonu tüm verilere uygulamak için kullanılır.
- **GetLogs**, veritabanında yapılmış olan tüm işlemlerin bilgisini kullanıcıya iletir.
- **ToSlice**, barındırılan verilerin bir kopyasını kullanıcıya iletir.

Ayrıca ORCA kullanılan verileri bileşik liste üzerinde saklar ve liste ile senkronize bir biçimde veritabanı değişikliklerini gerçekleştirir bu sayede kullanıcı büyük miktarda veri okumak istediğinde bu verilerin herhangi bir sistem çağrısı gerçekleştirilmeden ve veritabanıyla iletişim kurmak için bir soket açmaya gerek olmadan kısa bir süre içerisinde okunması istenen büyük miktarda veriyi kullanıcıya sunabilmektedir. Bu sayede veritabanı ve veritabanı sunucusu üzerindeki işlem yükünü de oldukça azaltmaktadır.

Bileşik liste ve üzerinde tanımlı metotlar ORCA'nın güncel versiyonun da desteklediği tüm veritabanları için ortaktır böylece yazılmış kod değiştirilmeden veritabanı değiştirilebilir. Burada veritabanı tipinin de bir önemi yoktur bu da aynı kodun desteklenen farklı veritabanları içinde çalışabilmesini ve taşınabilmesini sağlamaktadır.

2.3.4. Local Hooks

Kullanıcı ORCA'nın sağladığı bileşik liste üzerinde tanımlı metotları kullanarak veritabanı işlemlerini gerçekleştirebiliyor ancak bazı durumlarda kullanıcı kendisine sunulan bu metotların öncesinde ya da sonrasında çalışacak fonksiyonlar yazmak isteyebilir bu durumda ORCA kullanıcıya yerel kancaları(local hooks) sunar. Böylece kullanıcının tek bir yerde tanımladığı fonksiyon, bileşik listenin ilgili metodu her çağırıldığında ORCA tarafından çalıştırılacaktır ve böylece kullanıcı veritabanını yönettiği katmanı istediği gibi şekillendirme imkanına sahip olacaktır.

ORCA'da tanımlı yerel kancalar,

- **BeforeAdd**, bileşik listenin Add metodundan önce bir fonksiyon çalıştırılmasına imkan verir.
- **AfterAdd**, bileşik listenin Add metodundan sonra bir fonksiyon çalıştırılmasına imkan verir.
- **BeforeDelete**, bileşik listenin Delete metodundan önce bir fonksiyon çalıştırılmasına imkan verir.
- **AfterDelete**, bileşik listenin Delete metodundan sonra bir fonksiyon çalıştırmasına imkan verir.
- **BeforeUpdate**, bileşik listenin Update metodundan önce bir fonksiyon çalıştırılmasına imkan verir.
- **AfterUpdate**, bileşik listenin Update metodundan sonra bir fonksiyon çalıştırılmasına imkan verir.
- **BeforeAddRange**, bileşik listenin AddRange metodundan önce bir fonksiyon çalıştırılmasına imkan verir.
- **AfterAddRange**, bileşik listenin AddRange metodundan sonra bir fonksiyon çalıştırılmasına imkan verir.

Ayrıca kullanıcının tanımlamış olduğu fonksiyonlar, bileşik listenin ilgili metodunun içerisinde çalıştırılır böylece kullanıcı eğer isterse o an veritabanı işlemine katılan veriyi yazmış olduğu fonksiyon içerisinde kullanabilir. Örneğin, kullanıcı tanımladığı fonksiyonu BeforeAdd yerel kancasına atayabilir ve belirli kriterlere uyan bir veriyi veritabanına eklenmeden önce değiştirme imkanına sahip olabilir bu durumda veri değişmiş haliyle veritabanına eklenecektir.

Yerel kancaların her biri bir ID değerine ve önceliğe sahiptir böylece kullanıcı sınırsız sayıda fonksiyon tanımlayıp bir yerel kancaya atayabilir bu durumda tanımlanan fonksiyonlar kullanıcının belirlediği öncelik sırasına göre çalıştırılacaktır ayrıca kullanıcı belirli bir ID değerindeki yerel kancanın silinmesini de sağlayabilir.

Yerel kancalar kullanıcı tarafından kaynak kodda tanımlanır ve tanımlanmış oldukları kaynak kodda etkili olurlar.

2.3.5. Global Hooks

Global kancalar yerel kancalardan farklı olarak tanımlandıkları işletim sistemi üzerindeki aynı veritabanına aynı bağlantı cümlesi ile bağlanan tüm ORCA'ların bileşik liste metotlarını etkiler ve ayrıca yerel kancalar gibi kaynak kod içerisinde tanımlanma zorunlulukları yoktur.

Global kancalar yalnızca lider ORCA tarafından tanımlanabilir. Bir global kanca tanımlandığında lider ORCA tanımlanmış olan global kancayı diğer ORCA'lara Unix Soketlerini kullanarak iletir böylece global kanca aynı bağlantı cümlesini kullanan tüm ORCA'lar tarafından ortak olarak kullanılmaya başlar.

Global kancaların sağladığı bir diğer avantaj ise global kancanın kaynak kodda tanımlanma zorunluluğunun olmamasıdır. Global kanca Orca Language kurallarına uygun olarak karakter bazlı (string) tanımlanır ve bu tanım bir yorumlayıcı (interpreter) ile yorumlanarak programın çalışma zamanında kodun derlenmesine gerek duyulmadan çalıştırılır.

Go programlama dili derlenen bir dildir ve ORCA global kancalar sayesinde kullanıcıya kodu tekrar derlemeye gerek duymadan ya da değiştirmeden programa yeni fonksiyonellikler kazandırmasını sağlar.

2.3.6. Lider ORCA

Lider ORCA aynı işletim sistemi üzerinde bulunan ve aynı veritabanı bağlantı cümlesine sahip ORCA'lar arasındaki bağlantıyı sağlar ve global kanca tanımlama yetkisine sahip tek ORCA dır.

Kullanıcının lider ORCA tanımlamak için 1 satır kod yazması yeterli olmaktadır. Eğer aynı işletim sistemi üzerinde aynı veritabanı bağlantı cümlesine sahip başka ORCA'lar varsa birbirleri arasındaki iletişimi otomatik olarak sağlarlar kullanıcının burada herhangi bir şey yapmasına gerek duyulmaz.

Burada ORCA'lar arasındaki iletişim Unix soketler ile gerçekleştirilir. Örneğin, aynı veritabanı bağlantı cümlesine sahip ve aynı veri üzerinde değişiklik yapan iki ORCA yapılan değişiklikleri birbirlerine bu soketler üzerinden iletirler böylece veritabanı ve veritabanı sunucusu üzerine binen yük hafifletilmiş olur.

Kullanıcı eğer lider ORCA tanımını kaynak kodunda yapmazsa aynı işletim sistemi üzerinde aynı veritabanı bağlantı cümlesine sahip ORCA'lar bulunsa bile aralarında bir iletişim başlamayacaktır bu bakımdan kontrol ve kullanım kullanıcıya bırakılmıştır.

2.3.7. Optional Select / Auto Select

ORCA kullanıcıya veritabanındaki verileri okumak için iki farklı yöntem sunar.

Bu yöntemler,

- Otomatik select
- Opsiyonel select

ORCA varsayılan olarak veritabanındaki verileri okur ve daha hızlı yanıt verebilmek için bu verileri kendi ön belleğinde saklar. Bu sayede kullanıcı veritabanındaki verileri okumak istediğinde veritabanı üzerinde sorgu çalıştırmak yerine kendi belleğindeki barındırdığı verileri kullanarak kullanıcıya yanıt döner. Yani ORCA select işlemini kullanıcıdan bağımsız bir şekilde arka planda kendisi gerçekleştirir.

Eğer kullanıcı ORCA'nın kendi belleğinde verileri tutmasını istemiyor ya da veritabanından verileri kendi belirlediği şekilde okumak istiyorsa bu durumda ORCA kullanıcıya opsiyonel select imkanını tanır.

2.3.8. Generic Kolleksiyonlar

ORCA kullanıcıya elindeki verileri Go programlama dilini kullanarak filtreleyebilmesi ya da verilerin üzerinde çeşitli işlemler yapabilmesi amacı ile hazır listeler (koleksiyonlar) sunar böylece kullanıcı Go programlama dilinde performans kaygıları sebebiyle desteklenmeyen ancak diğer birçok dilde aşina olduğu koleksiyonları kullanma imkanına erişmiş olur. Koleksiyonların temel özelliklerini barındıran "container/list" paketi [13] kullanılarak günümüz koleksiyonlarının sahip olduğu fonksiyonlar geliştirilmiş ve ORCA'nın güncel versiyonunda kullanıcılara sunulmuştur.

ORCA kullanıcıya her biri farklı problemlerin çözümüne odaklanan dört farklı tipte liste sunmaktadır. ORCA'nın sunduğu bu araçlar tek bir satır kod ile birden fazla fonksiyonun, kullanıcının verileri üzerinde işlem yapmasını sağlayabilir böylece program geliştirme maliyetini en aza indirmek hedeflenir.

Mutable list, kullanıcıya veriler üzerinde işlem yapabilmesi için bazı metotlar sunar.Bu metotlar tanımlanan işlerini gerçekleştirmek için veri kaynağına veri kaynağının adresini kullanarak erişir ve veri kaynağını doğrudan değiştirirler. Bu durum kullanıcıya yüksek performans sağlarken veri kaynağına verinin adresi üzerinden erişildiğinden birden fazla iş parçacığının (goroutine) aynı anda veri kaynağına erişme ihtiyacı varsa, kullanıcı gerekli senkronizasyon işlemlerini yapmak durumundadır.

Immutable list, kullanıcının işlem yapmak istediği veri kaynağını kopyalar ve bu kopyayı kullanıcıya verir böylece aynı veri kaynağı birden fazla iş parçacığı (goroutine) tarafından senkronizasyona gerek duyulmadan kullanılabilir. Bu durumda veri kaynağı asla değişmez metotlar verinin kopyaları üzerinde işlem gerçekleştirir ve sonuç dönerler.

Lazy list, veri kaynağını değiştirecek işlemleri çağırılış sırasına göre saklar ve kullanıcı istediğinde bu işlemler gerçekleştirilir. Eğer kullanıcı çok büyük bir veri kaynağı üzerinde işlem gerçekleştirmek istiyor ancak bu işlemin sonucuna o an da ihtiyaç duymuyor ise lazy list kullanabilir böylece kullanıcı işlemci üzerindeki yükü kontrol etme imkanı bulmuş olur.

ORCA'nın sunduğu bu araçlar veritabanını asla etkilemezler amaçları kullanıcıya sorgu sonuçları üzerinde kolay bir şekilde işlem yapabilme imkanı sağlamaktır.

2.4. ORM Mimarisi

ORCA kullanıcıya veritabanındaki verilerini yönetebileceği bir koleksiyon sunar bu koleksiyon kullanılarak yapılan işlemler eğer veriyi değiştirmeye yönelik ise hem veritabanı hem de ORCA'nın kendi belleği senkronize bir şekilde değiştirilir ancak işlemler verinin okunmasına yönelik ise ORCA veritabanı tarafında bir sorgu çalıştırmadan gerekli bilgiyi kullanıcıya kendi belleğini kullanarak ulaştırır. Bu durum veritabanı sistemlerinin en büyük problemlerinden biri olan veri okuma hızını arttır ve ayrıca veritabanı üzerindeki işlem yükünü de azaltır.

Ayrıca ORCA aynı işletim sistemi üzerinde barındırılan uygulamaların veritabanı performansını arttırmaya yönelik de çözümler sunuyor. Aynı işletim sistemi üzerinde aynı veritabanı bağlantı cümlesine sahip olan ORCA'lar sürekli bir birleri ile iletişim içerisinde olarak hem kullanıcıya aynı işletim sistemi üzerinde barındırılan uygulamalar için toplu bir kontrol mekanizması sağlıyor hem de ilgili uygulamaların arasındaki iletişimi sağlayarak gerçek zamanlı veri paylaşımını kullanıcıya sunuyor.

ORCA aynı işletim sistemi üzerinde diğer ORCA kullanan uygulamalar arasındaki iletişimi Unix Soketlerini kullanarak gerçekleştiriyor. Unix Soketler TCP Soketlerinden daha az maliyetli ve hızlı olduğu için tercih ediliyor. Bu sayede veri de bir değişiklik yapıldığında değişikliği yapan ORCA diğerlerine değiştirdiği veriyi iletiyor böylece diğer ORCA'lar veritabanı tarafında bir sorgu çalıştırmadan değişikliği elde etmiş oluyor ayrıca bunu değişiklik gerçekleşir gerçekleşmez elde ediyorlar. Bu durum veritabanı üzerindeki işlem yükünü ciddi oranda azaltıyor. Örneğin, 10 uygulamanın ortak bir veritabanı kullandığını düşünelim ve veritabanına 1 kayıt girildiğini varsayalım önce veritabanı ile iletişim kurmak için bir TCP soketi oluşturulmalı ve her bir uygulamanın yapılan değişikliği alması için bir select sorgusu çalıştırması gerekli ve tabi ki veritabanının ilgili bilgileri cevap olarak dönebilmesi için disk üzerinde okuma yazma işlemleri yapması gerekecektir sonuç olarak 1 insert sorgusu, 9 select sorgusu, 10 tcp soketi ve 10 ayrı disk üzerine okuma ya da yazma işlemi gerçekleştirilmiş olur ancak eğer ORCA kullanılıyor olsaydı 1 insert sorgusu, 10 unix soketi, 1 TCP soketi ve 1 diske yazma işlemiyle sonuç elde edilmiş olacaktı. Bu durum ORCA kullanıldığında veritabanı ve veritabanı sunucusu üzerine binen işlem yükünün normalden 10 kat daha az olacağını göstermektedir.

ORCA kullanıcıya son olarak veritabanı işlemlerinden elde edilmiş veriler üzerine kullanıcı tanımlı ya da önceden tanımlanmış fonksiyonların kolayca uygulanmasını sağlayan araçlarda sunmuştur.

ORCA kendi belleği üzerinde işlem yaptığı verileri tutma yetisi sayesinde uygulama performansını arttırmayı hedeflemektedir.

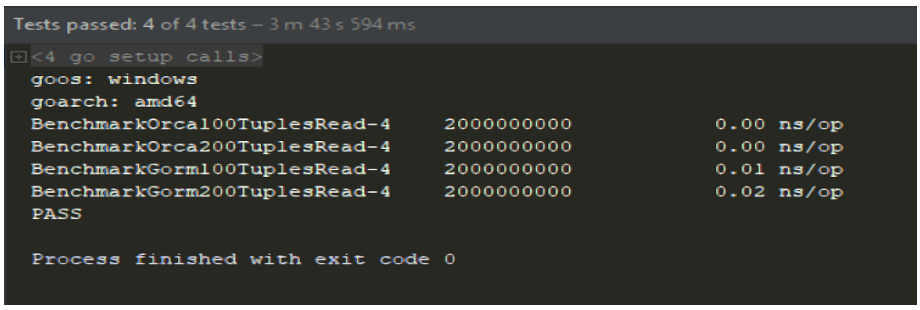
3. Araştırma Sonuçları ve Tartışma

3.1. Başarım Karşılaştırması

ORCA'nın diğer ORM (Object Relational Mapping) ve ODM (Object Document Mapping) araçlarından farkı kullanmakta olduğu verileri kendi belleğine alması bu sayede veritabanı tarafında çalıştırılan sorgu sayısını azaltarak hem veritabanı ve veritabanını bulunduran sunucunun işlem yükünü azaltmakta hem de kendi belleğinde tuttuğu veriler sayesinde kullanıcıya çok hızlı bir şekilde cevap verebilmekte.

Günümüzde Go programlama dilini kullanmakta olan geliştiriciler arasında oldukça popüler olan GORM ile geliştirmekte olduğumuz ORCA kullanılarak 4 farklı deney gerçekleştirilmiş ve iki kütüphanenin performansları karşılaştırılmıştır.

Test 1: Her iki kütüphane SQLite veritabanına önce 100 sonra 200 kayıt yazmış ve yazma işlemi bittikten sonra kayıtlar tümüyle okunmuştur. Her iki kütüphane her bir kayıt için birer transaction açıp kapatmıştır. Ölçüm işlemleri kütüphaneler verileri okumaya başladıklarında başlatılmıştır ve verileri veritabanına yazarken harcanan zaman ve kaynak sonuçlara dahil değildir.



Şekil 1. Test-1 sonuçları

Test-1'in sonuçlarını değerlendirecek olursak okunacak olan veri sayısını 2 kat arttırdığımızda GORM'un verileri okuma süresinin de 2 kat arttığını görmekteyiz lakin ORCA için durum böyle değil veri sayısının artmış olması ORCA'nın performansını etkilemiyor çünkü ORCA bu verileri veritabanına girerken aynı zamanda kendi belleğine de almıştı ve kullanıcı tüm verinin okunmasını istediğinde çok hızlı bir şekilde kullanıcıya yanıt dönebilmiştir.

Test 2: Her iki kütüphane SQLite veritabanına sürekli kayıt girmiş ve gerçekleştirilen her bir işlem için bellekte tahsis edilen bayt miktarı (B/op), işlem başına benzersiz bellek tahsisi işlemi sayısı (allocs/op) ve ilgili işlemin gerçekleştirilmesi için geçen süre ölçülmüştür.

```
C:\Users\Mehmet Emin\Desktop\ORCA Testler>go test -v -bench=. -benchmem
goos: windows
goarch: amd64
BenchmarkOrcaAdd-4            20         121061985 ns/op            3788 B/op        109 allocs/op
BenchmarkGormAdd-4            10         108457450 ns/op            9450 B/op        198 allocs/op
PASS
ok      _/C_/Users/Mehmet_Emin/Desktop/ORCA_Testler     5.073s
```

Şekil 2. Test-2 sonuçları

Test-2'nin sonuçlarını değerlendirecek olursak her iki kütüphane neredeyse aynı sürede işlemleri gerçekleştirmiş ancak ORCA'nın işlem başına bellekte tahsis edilen bayt miktarında GORM'dan 2,5 kat daha verimli olduğu ve işlem başına benzersiz bellek tahsisi işlemi sayısında ise GORM'dan 2 kat daha verimli olduğu görülmüştür.

Test 3: Her iki kütüphane SQLite veritabanında bulunan eşit miktar da ki verinin tamamını sürekli biçimde okudu ve sonuçları getirdi.

```
C:\Users\Mehmet Emin\Desktop\ORCA Testler>go test -v -bench=. -benchmem
goos: windows
goarch: amd64
BenchmarkOrcaTuplesReadSustanied-4       1000000              1993 ns/op            2032 B/op          7 allocs/op
BenchmarkGormTuplesReadSustanied-4           500           4793335 ns/op         1087836 B/op      21666 allocs/op
PASS
ok      _/C_/Users/Mehmet_Emin/Desktop/ORCA_Testler     55.711s
```

Şekil 3. Test-3 sonuçları

Test-3'ün sonuçlarını değerlendirecek olursak ORCA sonuçları yaklaşık 2500 kat daha hızlı bir şekilde kullanıcıya ulaştırmıştır. ORCA'nın işlem başına bellekte tahsis edilen bayt miktarında 535 kat daha verimli ve işlem başına bellekte tahsis edilen benzersiz bellek tahsisi işlemi sayısında 3000 kat daha verimli olduğu görülmüştür. Bu testte ORCA'nın vaat ettiği veri okuma performansı ciddi bir biçimde görülmüştür. ORCA'nın burada bu denli hızlı olmasının sebebi kullanıcıya beklediği yanıtı kendi belleğini kullanarak vermesindendir.

Test 4: Bu testte ORCA'nın MongoDB kayıt ekleme performansı ölçülmüştür. Her kayıt bir transaction altında veritabanına yazılmıştır.

```
C:\Users\Mehmet Emin\Desktop\ORCA Testler>go test -v -bench=. -benchmem
goos: windows
goarch: amd64
BenchmarkOrcaAddMongoDB-4          10000            134752 ns/op            4417 B/op         77 allocs/op
PASS
ok      _/C_/Users/Mehmet_Emin/Desktop/ORCA_Testler     221.017s
```

Şekil 4. Test-4 sonuçları

Test-4'ün sonuçlarını değerlendirecek olursak işlem başına bellekte tahsis edilen bayt miktarı ve işlem başına bellekte tahsis edilen benzersiz bellek tahsisi işlemi sayısında tatmin edici sonuçlar alınmıştır.

4. Sonuç

Veritabanı sistemleri içerisinde önbellek kullanımının veritabanı üzerindeki işlem yükünü ve veritabanını barındıran sunucunun disk işlemlerinden kaynaklanan işlem yükünü azalttığı bilinmektedir [14]. ORCA önbellek kullanımını ORM içerisine alarak önbellek mekanizmasına sahip olmayan bir veritabanının, önbellek mekanizmasını kullanan veritabanı kadar verimli sonuçlar almasını sağlayabilir.

Önbellek mekanizmasının veritabanı içerisinde değil de veritabanını kullanacak istemciler tarafından gerçekleştirilmesi de uzak sunucuda barındırılan bir veritabanı için sunucunun iletişim yükünün de büyük ölçüde ortadan kalkmasını sağlar [15].

Kaynakça

[1]Y. Ueda and M. Ohara, "Performance competitiveness of a statically compiled language for server-side Web applications," in *ISPASS 2017 - IEEE International Symposium on Performance Analysis of Systems and Software*, 2017, pp. 13–22.

[2]"blueperf/acmeair-monolithic-java: This version of Acme air is redesigned removing hardcoded components to WXS and also optimized for Cloud Data Services." [Online]. Available: https://github.com/blueperf/acmeair-monolithic-java. [Accessed: 03-Feb-2020].

[3]"blueperf/acmeair-monolithic-nodejs: A Node.js implementation of the Acme Air Sample Application. With datastore support of MongoDB. With runtime support of Bluemix/CloudFoundry, Docker... With Micro-Services." [Online]. Available: https://github.com/blueperf/acmeair-monolithic-nodejs. [Accessed: 03-Feb-2020].

[4]"index | TIOBE - The Software Quality Company." [Online]. Available: https://www.tiobe.com/tiobe-index/. [Accessed: 13-Feb-2020].

[5]"jinzhu/gorm: The fantastic ORM library for Golang, aims to be developer friendly." [Online]. Available: https://github.com/jinzhu/gorm. [Accessed: 03-Feb-2020].

[6]"xorm/xorm: Simple and Powerful ORM for Go, support mysql,postgres,tidb,sqlite3,mssql,oracle - xorm - Gitea: Git with a cup of tea." [Online]. Available: https://gitea.com/xorm/xorm. [Accessed: 03-Feb-2020].

[7]"src-d/go-kallax: Kallax is a PostgreSQL typesafe ORM for the Go language." [Online]. Available: https://github.com/src-d/go-kallax. [Accessed: 03-Feb-2020].

[8]"ezbuy/redis-orm: write db yaml once, generate go orm code everywhere." [Online]. Available: https://github.com/ezbuy/redis-orm. [Accessed: 03-Feb-2020].

[9]"zebresel-com/mongodm: A golang object document mapper (ODM) for MongoDB." [Online]. Available: https://github.com/zebresel-com/mongodm. [Accessed: 03-Feb-2020].

[10]"MehmetEminKaymaz/Orca: Simple and Powerful ORM for Go Language." [Online]. Available: https://github.com/MehmetEminKaymaz/Orca. [Accessed: 13-Feb-2020].

[11]"go-pg/pg: Golang ORM with focus on PostgreSQL features and performance." [Online]. Available: https://github.com/go-pg/pg. [Accessed: 13-Feb-2020].

[12]"PolyglotPersistence." [Online]. Available: https://martinfowler.com/bliki/PolyglotPersistence.html. [Accessed: 03-Feb-2020].

[13]"list - The Go Programming Language." [Online]. Available: https://golang.org/pkg/container/list/. [Accessed: 13-Feb-2020].

[14]K. Elhardt and R. Bayer, "A Database Cache for High Performance and Fast Restart in Database Systems," *ACM Trans. Database Syst.*, vol. 9, no. 4, pp. 503–525, Dec. 1984.

[15]R. Jain, Y. B. Lin, S. Mohan, and C. Lo, "A Caching Strategy to Reduce Network Impacts of PCS," *IEEE J. Sel. Areas Commun.*, vol. 12, no. 8, pp. 1434–1444, 1994.