T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BELLEK-İÇİ VERİ TABANLARI LİTERATÜR İNCELEMESİ

MEHMED TAHA ARAS

DOKTORA SEMİNERİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

DANIŞMAN
PROF. DR. HASAN HÜSEYİN BALIK

İSTANBUL, 2017

ÖNSÖZ

Bu çalışma boyunca bana ve çalışmamıza yaptığı rehberlikten ve katkılarından dolayı Prof. Dr. Hasan Hüseyin BALIK'a çok teşekkür ederim. Gerçekleştirilen bellek-içi veri tabanı incelemelerinde önerileriyle ve bilgi paylaşımlarıyla destek olan mühendis arkadaşlarıma ve desteklerini eksik etmeyen aileme teşekkürlerimi sunarım.

Mayıs, 2017

Mehmed Taha ARAS

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ vii

KISALTMA LİSTESİ viii

ŞEKİL LİSTESİ ix

ÇİZELGE LİSTESİ x

ÖZET xi

ABSTRACT xiii

BÖLÜM 1 1

GİRİŞ 1

1.1 Tezin Amacı ve Önemi ..... **Error! Bookmark not defined.**
1.2 Literatür Özeti 1
1.2.1 Literatürde Kullanılan Bellek-İçi Veri Tabanları 2
1.2.2 Literatürde Yer Alan İndeksleme ve Sorgulama Yaklaşımları 4
1.2.3 Literatürdeki Bellek-İçi Veri Tabanı Mimarileri 5

BÖLÜM 2 7

YÖTEMLER VE METODOLOJİLER 7

2.1 Bellek-İçi Veri Tabanı Yaklaşımları 7
2.1.1 Birincil Kullanım Bellek-İçi Veri Tabanı Yaklaşımı 8
2.1.2 İkincil Kullanım Bellek-İçi Veri Tabanı Yaklaşımı .. **Error! Bookmark not defined.**
2.2 Bellek-İçi Veri Tabanlarının Performans Optimizasyonları 9
2.2.1 Sorgulama Optimizasyonları 9

2.2.2 İndeksleme Optimizasyonları 10
2.2.3 Çoklu Thread Optimizasyonları 11
2.2.4 Büyük Veri Yönetimi İçin Bellek-İçi Veri Tabanı Performansı 12
2.3 Bellek Meşguliyetine Göre İş Yükü Dağılımı 13
2.4 Bellek-İçi Veri Tabanlarında Cluster Yönetimi 14
2.5 Dağıtık Bellek-İçi Veri Tabanı Yaklaşımı 14
2.6 Bellek-İçi Veri Tabanı İndeksleme Yöntemleri 15
2.6.1 Anahtar-Değer Tabanlı İndeksleme 15
2.6.2 Ağaç Tabanlı İndeksleme 16
2.6.3 Hibrit İndeksleme 16
BÖLÜM 3 18
GELECEK ÇALIŞMALAR 18
KAYNAKLAR 20
ÖZGEÇMİŞ 24

SİMGE LİSTESİ

I/O	Giriş/Çıkış

KISALTMA LİSTESİ

ACID	Atomicity, Consistency, Isolation, Durability
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
IBM	International Business Machines
NoSQL	Not only SQL
OLTP	Online Transaction Processing
RAC	Real Application Cluster
RAM	Random Access Memory
SAP	Systems, Applications and Products
SE	Societas Europaea
SQL	Structured Query Language

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖZET

BELLEK-İÇİ VERİ TABANLARI LİTERATÜR İNCELEMESİ

Mehmed Taha ARAS

Bilgisayar Mühendisliği Anabilim Dalı

Doktora Semineri

Danışman: Prof. Dr. Hasan Hüseyin BALIK

Günümüzde büyük ve kompleks verinin gerçek zamanlı olarak yoğun bir akış içinde işlenebilmesi ve sorgulanabilmesi ihtiyacı gittikçe daha fazla ortaya çıkmaktadır. Disk tabanlı veri tabanlarının kısıtlı I/O performansları bu ihtiyaçların karşılanmasın yetersiz kalabilmekte ve büyük çaplı uygulamalarda veri işleme noktasında darboğaza neden olabilmektedir. Disk tabanlı veri tabanlarının karşılaştığı bu problemi ortadan kaldırabilmek için ana bellek üzerinde verilerin depolanabildiği bellek-içi veri tabanları yaygınlaşmaya başlamıştır. Bellek-içi veri tabanları; belleğin çok yüksek hızlarda veri erişimine izin verdiğinden dolayı I/O darboğazının önüne geçmiş fakat belleklerin yüksek donanım maliyetinden dolayı hala büyük verilerin depolanmasında yetersiz kalmaktadır. Birçok farklı metodoloji ile tasarlanıp önerilen bellek-içi veri tabanı yaklaşımı bulunmaktadır.

Bu çalışmada literatürde önerilen bellek-içi veri tabanı yaklaşımları mimarilerine, sorgulama ve indeksleme metotlarına, eşzamanlı çalışma metotlarına, büyük veri işleme yöntemlerine, iş yükü dağıtım metotlarına, cluster yönetimlerine ve dağıtık çalışma mekanizmalarına göre incelenerek avantajları ve dezavantajları tartışılmıştır. Her bellek-içi veri tabanının kullandığı yöntemlere göre bazı kısıtları olduğu gözlemlenmiş ve çoğu çalışmada bu kısıtlar giderilebilmek için hibrit yaklaşımlara başvurulmuştur. Bellek-içi veri tabanlarındaki maliyetli donanım kısıtının minimize edilebilmesi için bellek optimizasyonunu amaçlayan tahminsel ve istatistiksel metotlar kullanılmıştır. Bellekte tutulan verinin akıllı bir şekilde seçilmesiyle erişilmek istenen

verinin bellekte bulunma ihtimali maksimize edilerek, olabildiğince çok sorgu ve manipülasyon işleminin belleğin yüksek hızından yararlanabilmesi amaçlanmıştır.

Anahtar Kelimeler: Bellek-içi veri tabanı, sorgulama, indeksleme, dağıtık mimari, büyük veri, cluster yönetimi, eşzamanlı çalışma, iş yükü dağıtımı, bellek optimizasyonu, tahminsel metotlar, istatistiksel metotlar

ABSTRACT

IN-MEMORY DATABASES LITERATURE REVIEW

Mehmed Taha ARAS

Department of Computer Engineering

PhD. Seminear

Adviser: Prof. Dr. Hasan Hüseyin BALIK

In present the need of real time processing and querying of big and complex data in an intensive stream has been increasing rapidly. Limited I/O performance of disk based databases can be insufficient to meet this need and they can cause a bottleneck in data processing in large scaled projects. In-memory databases are becoming more popular since they solve these problems which are faced by disk based databases. Because of that in-memory databases allow high speed access of data in memory; they have prevented the I/O bottleneck but they are still insufficient to store big data in terms of high memory costs.

In this study, we reviewed some main in-memory database approaches and discussed their advantages and disadvantages in terms of architectures, querying and indexing methods, concurrent working methods, big data processing methods, workload dispatching methods, cluster management and distributed working mechanisms. It has been observed that every in-memory database has its own constraints according to the methods they used. In order to prevent these constraints hybrid approaches have been preferred in most of studies. Inferential and statistical methods which aim to optimize the memory usage have been used to minimize the cost constraint of memories. Data which will be stored in memory are smartly determined to maximize the possibility of existence of queried data in memory. Thus it is aimed that as many queries and manipulation transactions as possible can benefit from the high speed of memories.

Keywords: In-memory database, querying, indexing, distributed architecture, big data, cluster management, concurrent working, workload placement, memory optimization, inferential methods, statistical methods

BÖLÜM 1

GİRİŞ

1.1 Seminerin Amacı ve Önemi

Verilerin işlenmesi ve saklanması süreçlerinde kullanılmak üzere kullanılan veri tabanı teknolojileri arasında bellek-içi veri tabanları giderek önemli konuma gelmiştir. Bu noktada bellek-içi veri tabanlarının popülaritesinin artmasında;

•Disk erişimlerindeki I/O yavaşlığı dar boğazının aşılabilmesi,

•Gerçek zamanlı ve büyük hacimdeki veri akışının isabetli ve performanslı bir şekilde kontrol altında tutulabilmesi,

•Belirli bir düzende akan verilerin tahminsel yöntemlerle tekrar tekrar diskten erişmek yerine bellekte tutulabilmesi ihtiyaçları rol oynamıştır.

Farklı özellikler ve karakteristikler sergileyen çok sayıda bellek-içi veri tabanı bulunmaktadır. Bu veri tabanlarının mimari açıdan, kullandıkları indeksleme ve sorgulama algoritmaları açısından ve optimal bellek kullanım yöntemleri açısından karşılaştırılması; hem en performanslı ana ve alt yöntemlerin belirlenmesi açısından hem de yeni bunların üzerine yeni yaklaşımların sunulabilmesi açısından büyük fayda sağlayacaktır.

1.2 Literatür Özeti

Bu bölümde bu çalışmalardan birincil, ikincil ve hibrit yöntemler izleyen farklı bellek-içi veri tabanı yaklaşımları ve teknikleri incelenecektir.

1.2.1 Literatürde Kullanılan Bellek-İçi Veri Tabanları

Kabakus[1] saklama türleri çerçevesinde bellek-içi veri tabanlarını “dokuman deposu”, “anahtar-değer deposu”, “kolon ailesi” ve “çizge veri tabanı” olmak üzere 4 kategoride incelemiş ve ilişkisel veri tabanı ile karşılaştırmıştır. Armstrong’un [2] aynı örnekler kullanılarak çizge veri tabanları ile diğer bellek-içi veri tabanlarının karşılaştırılamayacağı hipotezine dayanarak çizge veri tabanları bu karşılaştırmaya katılmamıştır. Yapılan deneyler sonucunda performanslar; (1) anahtar-değer çiftini yazma performansı, (2) verilen bir anahtar için değeri okuma performansı, (3) verilen bir anahtar için anahtar-değer çiftinin silinmesi performansı ve (4) tüm verinin çekilmesi performansı olmak üzere 4 kriterde ölçülmüştür. Yapılan her denemede üstünde çalışılan verinin büyüklüğü üstel olarak artırılmış bu sayede aynı zamanda veri büyüklüğündeki değişimlerin de performans üzerindeki etkisi gözlemlenmeye çalışılmıştır.

Kabakus’un çalışmasında dokuman deposu olarak MongoDB [3], Anahtar-değer deposu olarak Redis [4] ve Memcached [5], kolon deposu olarak Cassandra [6] ve ilişkisel veri tabanı olarak ise H2’yi incelemiştir. Çalışma sonucunda ise:

•Yazma işlemlerinde hız olarak en başarılı Memcached, bellek yönetimi olarak en başarılı ise Redis olduğu gözlemlenmiştir. MongoDB’nin veri büyüklüğü arttıkça performansının önemli ölçüde azaldığı görülmüştür.

•Okuma işlemlerinde Redis’in performansının Memcached ve MongoDB’den daha yüksek olduğu gözlemlenmiştir. H2 ise açıkça en düşük performansı göstermiştir.

•Silme işlemlerinde Redis ve Cassandra belleği çok daha efektif kullanmışlardır. Belleği en etkisiz kullanan H2 olmuştur.

•Tüm verinin getirilmesi işleminde MongoDB en iyi performansı gösterirken Cassandra ise en kötü performansı göstermiştir. Bellek kullanımında en efektif olarak yine MongoDB’nin olduğu gözlenmiştir. Veri büyüdükçe ise H2’nin çok yüksek oranlarda bellek kullanmıştır.

Oracle TimesTen Bellek-İçi Veritabanı[7] bellek en-iyilenmiş ilişkisel bir veri tabanıdır. Bu veri tabanı bağımsız tam işlevli bir veri tabanı olarak kullanılabileceği gibi aynı

zamanda bir Oracle İlişkisel Veri Tabanının işlem önbelleği olarak da kullanılabilmektedir. TimesTen tam-özellikli bir SQL desteğine ve aynı zamanda "Atomiklik", "Tutarlılık", "İzolasyon" ve "Süreklilik" (ACID) özelliklerine sahiptir. Uygulanan performans testleri sonucunda sahip olduğu depo yöneticisi sayesinde yüksek tutarlılığa sahip olduğu, çok-başlı replikasyon sistemi sayesinde erişilebilirliğe sahip olduğu görülmüştür. Ayrıca TimesTen performans testlerinde düşük tepki süresi ve esnek bir ölçeklenebilirlik sergilemiştir.

IBM SolidDB [8] bellek-içi veri tabanı çok yüksek seviyelerdeki hız ve erişilebilirlik özellikleri ile bilinmektedir. Bu veri tabanı esas olarak disk yerine ana belleğe dayalı olarak çalışmaktadır ve bu yaklaşım erişim hızının disk tabanlı veri tabanlarına göre çok daha yüksek olmasını sağlamaktadır. Bazı bellek-içi veri tabanları da bu seviyede yüksek erişim hızlarına ulaşabilmekte fakat büyük sayıda işlemlerle baş edememektedir. IBM SolidDB bu durumun aksine işlem kapasitelerinin yüksek ölçekli olduğu durumlarda bile çok düşük tepki sürelerinde çalışabilmektedir. SolidDB disklerin hız kısıtları ve belleklerin alan kısıtları arasındaki performans takası problemini çözebilmek için iki en iyilenmiş mekanizmadan oluşan hibrit bir yaklaşım kullanmaktadır. Bu mekanizmalar:

•Disk tabanlı erişimler için disk-tabanlı motor

•Bellek içi erişimler için ise ana-bellek motorudur.

Performans testleri sonucunda IBM SolidDB kullandığı hibrit mekanizma ile disk tabanlı veri tabanlarına göre 10 kat ve üzeri daha hızlı tepki süreleri sergilemiştir. Ayrıca içerdiği replikasyon sistemi ile de yüksek bir erişilebilirliğe sahiptir.

RAMCloud [9] büyük çaplı veri merkezi uygulamalarında kullanılan; dayanıklı veri depolarına düşük sürelerde erişimi mümkün kılmayı amaçlayan bellek-içi bir veri tabanıdır. RAMCloud düşük tepki sürelerine sahiptir, büyük verilere ölçeklenebilir, sürekliliği yüksektir, güçlü bir veri modeli vardır ve kolay bir şekilde sunucuya yüklenebilmektedir.

MemSQL[10] ayrık yapıda bellek-içi bir SQL veri tabanı yönetim sistemidir ve lişkisel veri tabanı yapısına sahiptir. Kilitsiz veri yapılarını ve tam-zamanında-işlemeyi (just-in-

time-compilation) beraber kullanarak yüksek istikrarsızlığa sahip iş yüklerini anlık olarak işleyebilmektedir.

SAP HANA [11] bellek-içi veri tabanı SAP SE tarafından geliştirilmiş, sütun-tabanlı, ilişkisel veri tabanı yönetim sistemidir. Süreçlerin karmaşıklıklarını düşürmek, içerdiği bulut mekanizması ile her ortamla entegre çalışabilmek ve iyileştirilmiş iş çıktıları SAP HANA veri tabanının hedeflediği anahtar faydalardır.

1.2.2 Literatürde Yer Alan İndeksleme ve Sorgulama Yaklaşımları

Bellek-içi veri tabanlarının indeksleme ve sorgulama mekanizmalarında çeşitli yöntem ve teknikler kullanılabilmektedir. Bu yöntemler sorgulama sürelerinin minimize edilmesini, istenilen indekse erişim sürelerinin minimize edilmesini ve indeks tablolarının bellekte kapladığı alan büyüklüğünü en aza indirmeyi amaçlamaktadır.

Ağaç tabanlı indeksleme bellek-içi veri tabanlarında kullanılan yaygın yöntemlerden bir tanesidir [12][13][14][15][16]. Yaprak düğümlerdeki değerlerin indeksin bellekteki yerini gösterdiği şekilde oluşturulan ağaçlar B-Ağacı, R-Ağacı ya da B+Ağacı gibi farklı algoritmalar ile oluşturulup kullanılırlar. Suei[17] bellek-içi veri tabanlarında kullanılacak yüksek lokal erişilebilirliği hedefleyen bir B+Ağacı tasarımı önermiştir. Aralıklı ve patlamalı olarak gelen dengesiz veri akışının kontrol altında tutulabilmesi için de bu yapıya ek olarak bir ağaç birleştirme algoritması kullanılmıştır. Bu yaklaşımın temel amaçları erişilmek istenen verinin olabildiğince bellekte bulunmasını sağlamak ve patlamalı olarak akan dengesiz veri akışının ağaç dengeleme işlemiyle kontrol altında tutulabilmesidir.

Suei[18] RFID uygulamalarında bellek-içi veri tabanı üzerinde etkili bir sorgulama ve depolamanın gerçekleştirilebilmesi için imza-tabanlı bir grid indeks yapısı önermiştir. Bu yaklaşımın temel amacı bellek-içi veri tabanlarında gerçekleşen veri manipülasyonlarında yüksek verimliliğin sağlanmasıdır. İmza olarak kullanılmak üzere veri parçasında seçilen özelliklerden oluşan bir küme işlenir ve bit vektörler olarak depolanır. Bu imzalar bir veri parçasının veri tabanında olup olmadığını etkili bir şekilde tanımlamaktadır.

Zhang[19] bellek-içi OLTP veri tabanlarında indekslerin bellekte büyük bir alan kaplamasını engellemek ve bellekte oluşacak ek yükü düşürmek amacı ile iki aşamalı bir indeks yaklaşımı önermiştir. Birinci aşamada gelen tüm girdiler alınır ve hızlı okuma ve yazma işlemleri için küçük tutulur. İkinci aşama daha kompakt ve okunması en iyilenmiş bir veri yapısına sahiptir. İndeks, girdileri periyodik olarak birinci aşamadan ikinci aşamaya taşımaktadır.

1.2.3 Literatürdeki Bellek-İçi Veri Tabanı Mimarileri

Günümüzde modern yazılım uygulamalarında kullanılan çeşitli bellek-içi veri tabanı mimarileri bulunmaktadır. Bu mimariler arasından, bellek-içi veri tabanlarının;

•Esas verinin erişilebilir olarak tutulmaya çalışıldığı

•Disk tabanlı bir veri tabanının ön bellekleme mekanizması olarak kullanıldığı

•Sıcak verilerin bellek-içi veri tabanında soğuk verilerin ise disk tabanlı veri tabanında tutulduğu

yapılar en yaygın olarak kullanılan mimarilerdir.

Esas olarak kullanılan ya da kullanılmak istenen verinin bellekte çok büyük bir erişilebilirlik olasılığına sahip olmasını amaçlayan yapılar bellek-içi veri tabanlarını birincil olarak kullanan yapılardır. Bu mimarilerde disk tabanlı veri tabanı verinin büyüklüğünün altından kalkabilmek için kullanılmaktadır. Ancak esas olarak erişilecek veri bellekte hazır olarak bulunmalıdır ve bunun gerçekleşmesi için farklı yöntemler ve algoritmalar geliştirilmiştir.

Disk tabanlı veri tabanlarının işlem performanslarını artırmak ve tepki sürelerini kısaltmak için bellek-içi veri tabanları ön bellekleme mekanizması olarak bir tampon gibi kullanılabilmektedir [20][21][22][23][24]. Bu tarz yaklaşımlarda bellek-içi veri tabanı esas veriyi üzerinde tutmaya çalışmaktansa üzerinde tuttuğu veriler sorgulandığında devreye girip disk-tabanlı veri tabanına zaman kazandırmayı amaçlamaktadır.

Bellek-içi veri tabanlarında sıcak ve soğuk verilerin tanımlanabilmesi bellek kullanımının optimize edilmesi açısından oldukça önemli bir işlemdir. Sıcak veriler sıklıkla erişilen

veriler temsil ederken soğuk veriler ise daha az ya da nadir olarak erişilen verileri temsil etmektedir. Bundan yola çıkarak soğuk verilerin bellekte tutulmasının bellekte gereksiz yük oluşturacağı kanısına varılmıştır. Böylece sıcak verilerin bellekte tutulup hızlı bir şekilde servis edilebilmesi, soğuk verilerin ise diskte tutulup gerektiğinde kullanmaya hazır olması sağlanmıştır. Sıcak ve soğuk verilerin tanımlanma işlemi sürekli kendini tekrarlamakta yani sıcak bir veri soğuk konuma ya da soğuk veri bir süre sonra sıcak veri haline gelebilmektedir. Bu hibrit yaklaşım veri tabanının ortalama performansını önemli ölçüde artırmaktadır [25][26][27][28][29].

BÖLÜM 2

YÖNTEMLER VE METODOLOJİLER

Bellek-içi veri tabanı kullanımı ve optimizasyonlarında çeşitli yöntemler öne sürülmüştür. Bu yöntemlerin çalışma modellerine ve süreçlerine göre kendi içlerinde farklı avantajları ve dezavantajları olduğu görülmüştür.

2.1 Bellek-İçi Veri Tabanı Yaklaşımları

Bellek-içi veri tabanları kullanıldıkları alana göre, kullanıldıkları sistemdeki kapatılmak istenen zayıf noktaya göre ve veri akışının karakteristiğine göre çeşitli yapılara sahip olabilirler. Sistemde veri akışı patlamalı ve dengesiz bir şekilde karakteristiğe sahipse tampon olarak kullanılabileceği gibi, bellekte istenen veriyi yüksek ihtimallerde barındırmaya yönelik daha merkezi bir konumda da kullanılabilirler. Bellek-içi veri tabanlarının sistemin hangi parçasında kullanılacağı ve nasıl işlev göreceğini belirlemek için temel olarak sistemin şu özelliklerini göz önünde bulundurmak gerekmektedir:

- Depolanması gereken veri hacmi
- Anlık işlem hacmi
- Bellek üzerinde tutulması istenen verinin karakteristiği
- Kullanılacak veri yapısına uygun indeksleme yöntemi

Bellek-içi veri tabanları kullandıkları verileri organize etme türlerine göre de değerlendirilirler. NoSQL bir veri tabanı olarak kullanılabildikleri gibi ilişkisel bir veri tabanı olarak da kullanılabilirler.

İlişkisel Bellek-İçi veri tabanlarında veri; geleneksel disk tabanlı ilişkisel veri tabanlarındaki gibi tablolar ve ilişkiler olarak organize edilir ve ACID özellikleri garanti edilir [30]. Ancak farklı olarak bu veri organizasyonu bellek üzerinde tutulmaktadır.

NoSQL veri depolama ve elde etme açısından ilişkisel veri tabanından farklı bir mekanizmaya sahiptir. NoSQL veri tabanlarında veri genellikle tablolu bir yapı yerine ağaç, çizge ya da anahtar-değer yapısına sahiptir. Ayrıca sorgulama dili olarak da SQL kullanılmaz. NoSQL veri tabanlarının ortaya çıkmasındaki ana düşünce ölçeklenebilir, erişilebilir ve daha basit bir yapıya sahip olmasıdır.

2.1.1 Birincil Kullanım Bellek-İçi Veri Tabanı Yaklaşımı

Bu yaklaşımda arkada bir disk ya da disk tabanlı veri tabanı kullanılmasına rağmen bellek-içi veri tabanları sistemdeki veri akışının merkezini oluşturmaktadır. Temel düşünce erişilmek istenen ya da beklenen verinin yüksek bir olasılıkla bellekte erişilebilir durumda hazır olmasıdır. Veri sorgulandığında bellekte hazır olacağı varsayıldığından bu sistemde yüksek oranlarda hız kazancı gözlemlenmiştir. Verinin bellekte hazır halde bulunması, indekslerin ve kayıtların bellekte minimum yer kaplaması ve iş yoğunluklarına göre sorgulamaların optimize edilmesi için çeşitli algoritma ve yöntemler kullanılmaktadır.

Bellekte esas erişilmek istenen verilerin yüksek erişilebilirlikte tutulabilmesinde en çok kullanılan yaklaşımlardan bir tanesi kayıtları ya da kolonları sıcak ve soğuk olarak kategorize etmektir. Sıcak olarak etiketlenen veriler bellekte tutulurken soğuk olarak etiketlenenler ise ikincil ve daha ucuz bir depolama alanına aktarılacaktır. Sıcak ve soğuk kayıtların etiketlenmesi işlemi çevrim dışı olarak ve periyodik gerçekleştirilir. Belirli bir zaman frekansında sorgulamaların tutulduğu log dosyaları analiz edilerek en çok sorgulanan kayıtlar ya da kolonlar tespit edilir [25][26]. Sıcak soğuk ayrımı için kullanılacak erişim sayısındaki sınır değeri hassas bir şekilde tespit edilmelidir. Çünkü sınır değerinin isabetsiz tespiti belleğin aşırı şişmesine ya da tam tersi olarak bellekte istenilen verilerin hazır olamamasına sebep olacaktır. Verinin anlık ve yoğun olarak aktığı sistemlerde loğlardan yapılan sorgulama analizi bellek optimizasyonunu geciktirebilmektedir. Anlık olarak yapılacak sıcak/soğuk veri etiketleme işlemi ise CPU üzerinde ek yük oluşturacaktır.

2.1.2 İkincil Kullanım Bellek-İçi Veri Tabanı Yaklaşımı

Bellek-içi veri tabanlarının avantajlarından faydalanabilmek için ve veri işlemlerinin etkinliğini ve ortalama hızını artırabilmek için bazı sistemlerde bellek-içi veri tabanları önbellekleme ya da tampon mekanizması olarak kullanılmıştır.

Uygulamanın veri kontrol mekanizması ile disk tabanlı veri tabanı arasında bellek-içi veri tabanı önbellekleme mekanizması olarak yer almaktadır. Uygulamadan herhangi bir veri erişim isteği olduğu zaman bu istek öncelikle önbellekleme mekanizmasına iletilir. Erişilmek istenen veri eğer önbellekte mevcut ise diske erişim olmadan istenen veri uygulamaya servis edilir. İlgili veri önbellekte mevcut değilse istek disk tabanlı veri tabanına iletilir ve diskten erişilen veri uygulamaya iletilir aynı zamanda da önbelleğe kaydedilir [20]. Önbellekleme mekanizmasının içeriği çok yüksek bir dinamikliğe sahip olacağından dolayı, erişilmek istenen olası verilerin mevcudiyetini yüksek oranlarda garanti edemez. Gerçek zamanlı ve yoğun veri akışı olan sistemlerde veriler kısmen daha yüksek olasılıklarla bellekten erişilir ve ortalama erişim süreleri kısalacağından performans artışı daha fazla olacaktır.

2.2 Bellek-İçi Veri Tabanlarının Performans Optimizasyonları

2.2.1 Sorgulama Optimizasyonları

Bellek-içi veri tabanlarında performans optimizasyonu yaparken uygulanan yöntemlerden bir tanesi sorgulama mekanizmaları üzerinde yapılan geliştirmelerdir. Veri elde etme, ekleme, silme ve güncelleme gibi işlemlerin etkinlikleri artırılıp tepki süreleri azaltılarak sistemin genel performansı artırılmaya çalışılır [31][32][33].

Sorgulama mekanizmasının performansını artırmak için kullanılan yöntemlerden bir tanesi paralel sorgulamadır. Sorguların bellek üzerindeki veriye paralel olarak erişmesi sağlanarak sıralı ve bekleyen işlemlerden kaçınılmaya çalışılır. Başarılı bir şekilde gerçekleşen paralel sorgular yoğun işlem hacmini kontrol altında tutmakta önemli rol oynamaktadır. Huang [34] CUDA platformu üzerinde paralel sorgulama sistemini kullanan bir sistem inşa etmiş ve bu sistemin performans sonuçlarını incelemiştir. Sistemden aldığın performans sonuçlarını MySQL'den aldıkları ile karşılaştırmıştır.

Paralel sorgulama mekanizması ile çalışan sistemin “join” operatörü başta olmak üzere tüm operatörlerde işlemleme süresinde azalma gözlemlenmiştir. Veri sıralama ve gruplama işlemleri işlem süresi açısından daha düşük seviyelerde olduğundan bu operatörlerde önemli bir performans artışı görülmemiştir.

Sorgulamaları kaynak kullanımına göre optimize ederek etkili kaynak yönetimini amaçlayan yöntemler de bulunmaktadır. Kraft [35] aynı anda gelen sorguların çatışmalarının ortaya çıkardığı negatif etkileri minimize etmek için sorguların organize edildiği bir yaklaşım önermiştir. Bu yaklaşımda bir sorgu kabul mekanizması bulunmaktadır. Bu mekanizma sorguların potansiyel olarak harcayacakları kaynağı hesaplayıp en az ve etkili kaynak kullanan sorguları önceliklendirir. Önceliklendirme sıralarına göre sorguları kabul ederek işleme alınmalarını sağlar. Bu mekanizma herhangi bir t anındaki kaynak kullanımını optimize etmeyi amaçlamaktadır. Sorguların potansiyel kaynak kullanımları; sistemin o anki davranışları ile ideal durumda sergileyeceği davranışların karşılaştırılması ile elde edilen bir metrik aracılığı ile hesaplanmaktadır. Bu metrik aynı zamanda hangi sorgunun sistem kaynaklarını beklenenden daha etkili kullandığının tespit edilmesini de sağlamaktadır.

2.2.2 İndeksleme Optimizasyonları

Veri tabanlarında istenilen veriye en hızlı sürede ve en etkili şekilde erişebilmek için indeksleme sistemleri kullanılmaktadır. Bellek-İçi veri tabanlarında kaynak kullanımını ve indekslerin bellekte kapladıkları alanı optimize etmek içi çeşitli yöntemler bulunmaktadır.

Suei’nin [18] önerdiği grid indeks yapısı bir tablonun ya da ilişkinin anahtarları olan çok boyutlu özelliklerden oluşmaktadır. İstenen bir parça verinin değeri o veri parçası için önceden belirlenen bir hash fonksiyonu ile elde edilir. Bu yaklaşımda birbirine bağlı grid düğümleri içinde depolanan, aynı anahtara sahip verinin aynı grid hücresinde yer aldığı bir veri organizasyonu önerilmiştir. Her bir grid hücresi düğümlerden oluşan bir liste içerir ve her düğüm bir sayaç, bir imza ve bir diziden oluşur. Sayaç; düğümde depolanan kayıtların sayısını tutmaktadır. İmza ise düğümde depolanan verilerin özetidir ve arama işlemini hızlandırmak amacıyla kullanılır. Bir veri kaydının imza yapısı

düğümde depolanan kayıtların seçilen özelliklerindeki seçilen bitlerinin birleşmesinden oluşur.

Ağaç indeks yapısı da bellek-içi veri tabanlarında kullanılan yaygın bir indeksleme yöntemidir. Bu ağaç yapılarından en yaygın olarak kullanılanlardan bir tanesi B-Ağacıdır. Bu ağaç yapısı logaritmik sürede arama, sıralı erişim, ekleme ve silme işlemlerini gerçekleştirebilmek için kendini yeniden dengeleyebilen ve bu sayede veriyi belirli bir sırada tutabilen bir iç mekanizmaya sahiptir. İkili arama ağacının genelleştirilmiş bir halidir, farklı olarak bir düğüm ikiden fazla çocuğa sahip olabilir. Ayrıca ikili arama ağacının aksine büyük bloklar halindeki verinin okunup yazılabilmesi için optimize edilmiştir. B+ ağacı da bellek üzerinde indekslemelerde kullanılan başka bir yapıdır. Çoğunlukla B-Ağacına benzeyen bu yapının farkı düğümlerde sadece anahtar içermesi ve en alt seviyesine birbirine bağlı yapraklar eklenmesidir. Düğümlerde yalnızca anahtarları tutmasından dolayı bellekteki bir sayfaya daha çok anahtar sığabilmektedir.Ayrıca B+ ağacı üzerinde tam tarama yapmak daha maliyetsiz olacaktır.

2.2.3 Çoklu Thread Optimizasyonları

Bellek-içi veri tabanlarında iş yükünü bellek kümelerine optimal şekilde dağıtabilmek için tahmin yöntemleri kullanılmaktadır. Gelen sorgudaki iş yükünün ne olacağı tahmin edilip bellekteki dağıtım paralel threadler kullanılarak gerçekleştirilir [23][36][37][38]. Threadler üzerinden gerçekleşen dağıtım iş yüklerinin karakteristiklerine göre farklı algoritmalar kullanılarak sağlanabilmektedir. Büyük hacimli işler, uzun sürmesi beklenen işler ya da küçük iş parçacıkları gibi kategorize edilecek olursa dağıtım ve işleme sıraları tamamen bellek-içi veri tabanının sahip olduğu algoritmalara bağlı olacaktır.

Bellek-içi veri tabanlarında çoklu threadleme yöntemi yazılım performansının artıracağı gibi donanım performansını da artıracaktır. Bellek-içi veri tabanlarında çoklu threadleme kullanıldığında tek thread ile çalışmasına kıyasla %30 ile %70 arasında performans artışı olduğu gözlemlenmiştir[39]. Meyer [40] ise çalışmasında bunun aksine herhangi bir t anında paralel olarak çalıştırılan işlerde disk tabanlı veri tabanlarının daha yüksek performans gösterdiğini savunmuştur.

2.2.4 Büyük Veri Yönetimi İçin Bellek-içi Veri Tabanı Performansı

Ana belleklerin kapasitelerinin yükselmesi büyük verilerin bellek-içi veri tabanlarında yönetilmesi ve işlenmesine olanak sağlamıştır. Büyük veriler bellek-içi veri tabanlarında yönetilirken can alıcı noktalar şu şekildedir[30]:

•İndekslerin kullanımı geleneksel disk tabanlı veri tabanlarında olduğundan çok daha farklıdır. İndeks kullanımlarının sadece I/O hızına odaklandığı disk tabanlı yaklaşımların aksine bellek ve önbellek kullanımının optimize edilmesi gerekmektedir[41][42]. Memcached[5], Redis[4] ve RAMCloud[9]'da olduğu gibi genellikle hash-tabanlı indeksler daha çok tercih edilmektedir.

•Veri düzeninin de bellek kullanımı ve önbellek optimizasyonunda önemli etkisi bulunmaktadır. İlişkisel tabloların kolonlu yapısı tarama tarzı analitik ve sorgularda kolaylık sağlarken, satır bazlı çalışmaya daha uygun olan OLTP sorgularında düşük performans gösterebilmektedir.

•Yük dağılımını dengeleyebilmek için paralelizm özelliğinden faydalanmak gerekmektedir. Paralelizm CPU içinde tek bir döngüde yapılacak işlerin birleştirilmesi şeklinde olabileceği gibi, çok çekirdekli paralel yapı kullanılarak aynı anda birden fazla CPU çekirdeğinin işlem yapması şeklinde de olabilmektedir.

•Aynı anda yapılması gereken işlemlerin etkili şekilde yönetilebilmesi gerekmektedir. Büyük sistemlerde tutarlığın sağlanabilmesi için kilit mekanizmaları ve çıkmaz kontrol yöntemleri kullanmak performansı önemli ölçüde düşürebilmektedir. Bunun yerine önceden tanımlanmış zaman sıralarında işlemlerin gerçekleştirilmesi daha optimize bir çözüm olacaktır.

•Sorguların işlenme yöntemleri performans üzerinde olumlu ya da olumsuz büyük etkiler gösterebilmektedir.

•Yazılım, donanım ya da enerji faktörlerinden kaynaklanabilecek olası veri kayıplarının ve tutarsızlıkların önlenebilmesi için hata toleransının yüksek olması gerekmektedir.

•Belleğe sığmayan büyük ölçekli verinin akıllı bir şekilde alternatif bir depolama birimine aktarılabilmesi gerekmektedir. Günümüzde bellek kapasiteleri çok daha büyük olmasına rağmen büyük veri ile uğraşırken bu tarz taşmaların yaşanması çok normaldir.

Bu tarz durumlarda sıklıkla uygulanan yöntemlerden bir tanesi sık erişilen verileri bellekte tutmaya devam ederken, seyrek erişilen verileri alternatif depolama alanına aktarmaktır.

2.3 Bellek Meşguliyetine Göre İş Yükü Dağılımı

Bellek-içi veri tabanlarında anlık olarak gelen iş yükleri dengeli bir şekilde belleğe dağıtılmalı ve bellek kullanımı optimize edilmelidir. Bu optimizasyonu gerçekleştirebilmek için tahmin veya anlık iş yükü analizinden faydalanan gönderici mekanizmalar kullanılmıştır. Tahmine dayalı iş yükü dağıtım mekanizmaları sıklıkla kullanılmasına karşı çok fazla değişkenlik gösteren paralel çalışma seviyeleri başarılı tahminlerin yapılabilmesini engelleyici önemli bir faktördür. Bu sorunu aşmak için genellikle threadleme seviyelerini de göz önünde bulunduran tahmine dayalı hibrit yöntemlere başvurulmuştur.

Molka[36] çalışmasında büyük verilerin yönetildiği bellek-içi veri tabanlarında iş yükü dağıtımının etkili bir şekilde optimizasyonu için bir performans modellemesi yapmıştır. Özellikle analitik iş yükleri altında çeşitli threadleme seviyelerine ve sınıf başına bellek meşguliyetini modellemek için fork-join (paralel) sorgulama tabanlı bir yöntemle yanıt sürelerinin tahminlerini ve uyuşmazlık oranlarını kullanmayı önermişlerdir. Bu çalışmanın yapı taşları şu şekildedir:

•Thread seviyesindeki paralelliği ve uyuşmazlık ihtimallerini göz önünde bulundurarak yanıt sürelerinin tahmin edilmesi

•Bellek-içi veri tabanın clusterları için performansı ve maliyeti optimize etmek için yük dağıtım olasılıklarının aranması

•Önerilen yöntemin büyük çaplı sistemlerde kullanılabilirliğinin doğrulanması

Molka[43] bir diğer çalışmasında analitik ve olasılıksal modelleri birleştirerek hibrit bir bellek kullanım optimizasyonu önermiştir. Bu çalışmada sorguların büyüklükleri ile bellekte işgal ettikleri alanlar ilişkilendirilerek henüz sorgu işlenmeden ilgili tahminler yapılıp iş yükü dağıtımı da bu tahminlere göre yapılmıştır. Yapılan tahminlerin ortalama 1% ile 5% arasında hata oranı gösterdiği ve maksimum olarak 25% hata oranının geçilmediği görülmüştür. Hibrit yapı kullanmayan tahminsel yöntemler ise düşük bir

performans göstererek, ortalama 20% oranından yüksek hata oranlarına sahip olabilmektedir.

Verma[44] bellek-içi veri tabanlarında sözlük tablolarındaki kullanılmayan alanların istatistiksel bir metrik baz alınarak sıkıştırılması ve bu sayede bellek kullanımının optimize edilebileceğini önermiştir. Bu yöntemde sütun bazlı bir indirgenme yapılmıştır. Satır tanımlayıcılarının kaybolmayacağı, direk erişimin kısıtlanmayacağı ve CPU yükünün artmayacağı savunulmuştur. Bu çalışmada sözlük tabloları delta şifrelemesi metodunu kullanarak sıkıştırılmıştır.

2.4 Bellek-İçi Veri Tabanlarında Cluster Yönetimi

Bellek-içi veri tabanlarında verinin depolandığı ana bellek küme(cluster)'lerden oluşmaktadır. Verinin bu clusterlar arasında dengesiz dağılımın bellek optimizasyonunu olumsuz olarak etkileyecektir. Gelen yükün rastgele dağıtımı durumunda bazı clusterlar boş olarak beklerken bazı clusterlarda ise yük aşımı olabilmektedir. Cluster'ların optimize şekilde kullanılabilmesi için verinin yerleştirilmesinde bir otonom mekanizmanın bulunması gerekir.

Schaffner [45] diğer çalışmaların aksine sorguların tekil maliyetlerini karakterize etmek yerine, anlık sorgu akışını, önceden belirlenen yanıt süre hedeflerine göre karakterize etmiştir. Bir sunucunun belirli bir yanıt süresinin aşmamak kaydı ile ne kadar yükü kaldırabileceği tahmin edilmiş ve bu sunucuya bu tahminlere göre dağıtım yapılmıştır. Sunuculara atanan kiracıların tepki sürelerine zararı olduğu tahmin edildiği an kiracıların başka uygun bir sunucuya atanması sağlanmıştır. Gerekli durumlarda sunucular arası kiracı geçişi sağlanarak sunucu topluluğundaki yük dengesi stabil tutulmuştur. Bu adımların uygulanması ile sunucular ya da clusterlar arasındaki yük dengesi sağlanmaya çalışılmıştır.

2.5 Dağıtık Bellek-İçi Veri Tabanı Yaklaşımı

Bellek-içi veri tabanlarında yönetilen verinin karakteristiğinin değişmesinden, iş alanının genişlemesinden ya da başka genişleyici etkenlerden dolayı veri tabanının ölçeklenebilir olması önemli bir gereksinimdir. Bellek-içi veri tabanlarının büyük

ölçeklere taşınabilmesi için dağıtık bir bellek mimari kullanılması günümüz teknolojisinde olmazsa olmaz bir faktördür.

Oracle Real Application Cluster (RAC) kullanıcılarına veri tabanı sunucuları kümesinin yapılandırılabilmesini sağlamaktadır. Herhangi bir Oracle objesinin verisi paylaşılan bir depolama mekanizmasında "Oracle Veri Blokları" şeklinde saklanmaktadır. Bu veri blokları paylaşılan bir "veri tabanı tampon önbelleğinden" erişilebilmektedir. Bellekte kalması istenen bir objenin bellek üzerindeki devamlılığından "dağıtım yöneticisi" sorumlu olmaktadır [46].

Iwazume [47] çalışmasında veri tabanının durdurulmasına gerek kalmadan verinin ve baş düğümlerin eklenebildiği dağıtık ve ölçeklenebilir bir bellek-içi veri tabanı yaklaşımı önermiştir. Çok büyük ölçeklerde veri için bir altyapı oluşturulmaya çalışılmıştır. Okuyama adını verdikleri bu bellek-içi veri tabanında ayrık algoritma olarak "Tutarlı Hashleme" metodu kullanılmıştır. Veri düğümleri eklendikçe veri bozulmasını engellemek için veri yapısı özgün bir şekilde tekrar düzenlenmektedir. Okuyama hem disk tabanlı hem de bellek tabanlı çalışabildiği gibi, belleği sadece verilerin anahtarlarını tutmak için de kullanabilmektedir. Bu çalışmada olduğu gibi SAP HANA [48] veri tabanı sisteminde de düğümlerin eklenmesi ile elde edilen dağıtık ve ölçeklenebilir bir mimari bulunmaktadır.

2.6 Bellek Meşguliyetine Göre İş Yükü Dağılımı

Bu bölümde bellek-içi veri tabanlarında kullanılan indeksleme yaklaşımlarından bahsedilecektir.

2.6.1 Anahtar Değer Tabanlı İndeksleme

Bellek-içi veri tabanlarında en çok kullanılan indeksleme yöntemlerinden birisi anahtar-değer tabanlı indeksleme sistemidir. Veri tabanında bulunan düğümler üzerinde anahtar ve değer kısımları yer alır. Her bir veri objesinin eşsiz bir anahtara ve içeriğini oluşturan bir değere sahip olması gerekir. Anahtarı bilinen bir veri objesine erişmek çok düşük tepki süreleriyle gerçekleşecektir.

Veri objelerinin anahtar değerleri kullanıcıların tarafından verilebileceği [20][49] gibi anahtar-değer tabanlı indekslerin üzerine geliştirilen ve anahtar yerine veri özelliklerinden oluşan bir imza yapısı kullanan sistemler [18][50] de bulunmaktadır.

2.6.2 Ağaç Tabanlı İndeksleme

Özellikle ilişkisel veri tabanlarında yoğunlukla kullanılan indeksleme mekanizması ağaç-tabanlı bir yapıya sahiptir. Verinin indekslenmesi amaçlanan özellikleri arka planda belirlenen bir ağaç algoritmasına göre bir ağaç formunda bulunur. Bu veri objesinin indekslenmiş herhangi bir özelliğine göre sorgulama yapıldığında önceden oluşturulmuş bu ağaç yapısı ve ağaç algoritması kullanılarak verinin bellek veya disk üzerindeki konumu bulunur. Bulunan konumdan getirilen veri sorgulayan uygulamaya servis edilir.Bu indeksleme yönteminde B-Ağacı, B+Ağacı ve R-Ağacı en çok kullanılan yapılardır. Ayrıca 2-3-4 Ağacı, UB Ağacı ve Bx Ağacı indeksleme mekanizmalarında kullanılabilen yapılardır[17].

Bellek-içi veri tabanlarındaki ağaç indeks yapıları düğümler ve yapraklardan oluşmaktadır. Düğümler bellekte bağlantılı olduğu düğümlerin ve yaprakların bilgilerini barındırırken, yapraklar ise aranan verinin bulunduğu bellek ya da disk gözünün adresini içerir.

Suei [17] çalışmasında bellek-içi veri tabanlarında dengesiz büyüklükte gelen veri yapılarını kontrol edebilmek ve tampon taşmalarını engelleyebilmek için B+ Ağacı yöntemini kullanmıştır. Büyük veri yapıları ile çalışırken tek seferde işlememiz gereken verinin yapısı bazı durumlarda bellekte bulunan tamponların boyutunu aşabilmektedir. Bu olumsuz durumların önüne geçebilmek için B+ Ağacının başarılı birleştirme algoritması kullanılmıştır.

2.6.3 Hibrit İndeksleme

İndeksleme yöntemlerinin zayıf yönlerini olabildiğince minimize edebilmek için birden fazla indeksleme yönteminin beraber kullanıldığı yöntemlerdir.

Zhang [19] OLTP veri tabanlarında ana bellekteki depolama fazlalığını düşürebilmek için hibrit bir çözüm önermiştir. Bu çözüm iki aşamadan oluşmaktadır:

•Birinci aşama dinamik aşamadır. En son erişilen eklenen girdilerin hızlı erişilmesi ve veri tabanına yazılmasını sağlamak için dinamik bir veri yapısı yazma tamponu olarak kullanılır.

•İkinci aşama statik aşamadır. Soğuk verinin okuma işlemleri için kompakt ve okuma-optimal bir veri yapısı kullanılır.

Bu hibrit yapının kullanımında diğer indeksleme yöntemleri ile karşılaştırıldığında aynı verimi daha düşük bellek kullanımı ile sağlayabildiği görülmüştür.

BÖLÜM 3

GELECEK ÇALIŞMALAR

Günümüzde mevcut olarak kullanılan bellek-içi veri tabanlarının depolama alanı kısıtlarından dolayı hala verinin sadece belli bir bölümü bellek üzerinde tutulabilmektedir. Verinin geri kalan büyük bir kısmı ise disk üzerinde tutulmaktadır. Tahminsel yöntemler ve erişim istatistiklerini kullanan yöntemler bellek alanını optimum şekilde kullanmayı amaçlasa da tam anlamıyla başarılı olamamaktadırlar. Bu yüzden gelecek çalışmalarda verinin daha büyük kısmını bellekte tutulabilmesini sağlayabilecek etkili sıkıştırma teknolojileri ve veriyi en büyük ölçüde temsil edebilecek veri özeti çıkarma yöntemleri kullanılmalıdır. Bu sayede yaygın olarak kullanılan hibrit yöntemlerin yerine bellek ağırlıklı depolamanın olduğu sistemler geliştirilebilecektir.

Bellek-içi veri tabanlarında yapılan sorguların ön tahminleri genellikle hataya açık olmaktadır. Belirli bir standarda dayanmayan ölçümler sorgudan sorguya değişebilmekte ve bazı sorgularda doğruya çok yakın tahminler üretirken bazılarında ise yanlış tahminler yapılmaktadır. Kullanılan sorguların olası bellek ve işlemci meşguliyetini daha kesin bir şekilde tespit etmek, yanlış iş dağıtımlarının önüne geçerek bellek ve işlemcideki optimizasyonu güçlendirecektir. Bu süreci şu an literatürde çoğunlukla kullanılan tahminsel yapıdan daha kararlı bir tespit mekanizmasına çevirebilmek için bellek-içi veri tabanlarında kullanılan sorguların standardize edilerek bellek ve işlemci kullanımını net bir şekilde ölçebilecek forma dönüştürülmesi önemli bir çalışma alanı olacaktır.

Bellek-içi veri tabanlarında sunucuların, bu sunuculardaki clusterların ve hatta bu clusterlardaki düğümlerin kullanılabilirlik yüzdeleri bütüncül bir yaklaşımla devamlı

olarak izlenebilmesi ve işlerin maksimum düzeyde homojen olarak dağıtılabilmesi için bir mekanizma geliştirilmesi de gelecekte üzerinde durulması gereken konulardandır.

KAYNAKLAR

[1] Kabakus, Abdullah Talha, and Resul Kara. "A performance evaluation of in-memory databases." Journal of King Saud University-Computer and Information Sciences (2016).

[2] Armstrong, T.G., Ponnekanti, V., Borthakur, D., Callaghan, M., 2013. LinkBench: a database benchmark based on the Facebook social graph. In: SIGMOD ’13 Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, New York, NY, USA, pp. 1185–1196. http://dx.doi.org/ 10.1145/2463676.2465296.

[3] MongoDB Inc. “MongoDB Architecture Guide,” 2015.

[4] S. Sanfilippo and P. Noordhuis. (2009). Redis [Online]. Available: http://redis.io

[5] B. Fitzpatrick and A. Vorobey. (2003). Memcached: A distributed memory object caching system [Online]. Available: http://memcached.org/

[6] DataStax. “Introduction to Apache Cassandra,” 2014.

[7] Lahiri, Tirthankar, Marie-Anne Neimat, and Steve Folkman. "Oracle TimesTen: An In-Memory Database for Enterprise Applications." IEEE Data Eng. Bull. 36.2 (2013): 6-13.

[8] Lindström, Jan, et al. "IBM solidDB: In-Memory Database Optimized for Extreme Speed and Availability." IEEE Data Eng. Bull. 36.2 (2013): 14-20.

[9] S. M. Rumble, A. Kejriwal, and J. Ousterhout, “Log-structured memory for dram-based storage,” in Proc. 12th USENIX Conf. File Storage Technol., 2014, pp. 1–16.

[10] G. Nikolova and T. Ha. “In-Memory Databases MemSQL,” IT4BI - Université Libre de Bruxelles, 2015.

[11] F. Färber, et al. “The SAP HANA Database – An Architecture Overview,” IEEE Computer Society Technical Committee on Data Engineering, pp. 28-33, 2012.

[12] Leis, Viktor, Alfons Kemper, and Thomas Neumann. "The adaptive radix tree: ARTful indexing for main-memory databases." Data Engineering (ICDE), 2013 IEEE 29th International Conference on. IEEE, 2013.

[13] Cui, Bin, et al. "Main memory indexing: the case for BD-tree." IEEE Transactions on Knowledge and Data Engineering 16.7 (2004): 870-874.

[14] Kchoi, K. R., and KC KIM. "T∗-tree: a main memory database index structure for real time applications [C]." RealTime Computing Systems and Applications (1996).

[15] Lu, Hongjun, Yuet Yeung Ng, and Zengping Tian. "T-tree or b-tree: Main memory database index structure revisited." Database Conference, 2000. ADC 2000. Proceedings. 11th Australasian. IEEE, 2000.

[16] Li, Lu, et al. "Efficient Tree Indexing for PCM-Based Memory Systems." Control and Automation (CA), 2015 8th International Conference on. IEEE, 2015.

[17] Suei, Pei-Lun, et al. "An efficient B+-tree design for main-memory database systems with strong access locality." Information Sciences 232 (2013): 325-345.

[18] Suei, Pei-Lun, et al. "A signature-based Grid index design for main-memory RFID database applications." Journal of Systems and Software 85.5 (2012): 1205-1212.

[19] Zhang, Huanchen, et al. "Reducing the storage overhead of main-memory OLTP databases with hybrid indexes." Proceedings of the 2016 International Conference on Management of Data. ACM, 2016.

[20] Liu, Chengjian, et al. "R-memcached: a reliable in-memory cache for big key-value stores." Tsinghua Science and Technology 20.6 (2015): 560-573.

[21] Hu, Xiameng, et al. "Optimized Locality-aware Memory Management in Key-value Cache." IEEE Transactions on Computers (2016).

[22] Ou, Jianqiang, et al. "A penalty aware memory allocation scheme for key-value cache." Parallel Processing (ICPP), 2015 44th International Conference on. IEEE, 2015.

[23] Ma, Kun, and Bo Yang. "Access-aware in-memory data cache middleware for relational databases." High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on. IEEE, 2015.

[24] Patil, Anil, and Rajesh Ingle. "Leveraging Information Bus with In-Memory Caching for Service Oriented Architecture." Advances in Computing and Communications (ICACC), 2013 Third International Conference on. IEEE, 2013.

[25] Levandoski, Justin J., Per-Åke Larson, and Radu Stoica. "Identifying hot and cold data in main-memory databases." Data Engineering (ICDE), 2013 IEEE 29th International Conference on. IEEE, 2013.

[26] Afify, Ghada M., Ali El Bastawissy, and Osman M. Hegazy. "A hybrid filtering approach for storage optimization in main-memory cloud database." Egyptian Informatics Journal 16.3 (2015): 329-337.

[27] Chandramouli, Badrish, Justin Levandoski, and Eli Cortez. "ICE: Managing cold state for big data applications." Data Engineering (ICDE), 2016 IEEE 32nd International Conference on. IEEE, 2016.

[28] Li, Yongkun, et al. "Workload-aware Elastic Striping with Hot Data Identification for SSD RAID Arrays." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2016).

[29] Kobayashi, Atsuro, Tsukasa Tokutomi, and Ken Takeuchi. "Versatile TLC NAND flash memory control to reduce read disturb errors by 85% and extend read cycles by 6.7-times of Read-Hot and Cold data for cloud data centers." VLSI Circuits (VLSI-Circuits), 2016 IEEE Symposium on. IEEE, 2016.

[30] Zhang, Hao, et al. "In-memory big data management and processing: A survey." IEEE Transactions on Knowledge and Data Engineering 27.7 (2015): 1920-1948.

[31] Playfair, Daniel, et al. "Big data availability: Selective partial checkpointing for in-memory database queries." Big Data (Big Data), 2016 IEEE International Conference on. IEEE, 2016.

[32] Alam, Maksudul, Srikanth B. Yoginath, and Kalyan S. Perumalla. “Performance of Point and Range Queries for In-memory Databases using Radix Trees on GPUs.” Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), 2016.

[33] Teshome, Surafel, and Tae-Sun Chung. "Query cost estimation for read intensive flash memory based database systems." Electronics and Information Engineering (ICEIE), 2010 International Conference On. Vol. 1. IEEE, 2010.

[34] Huang, Yin-Fu, and Wei-Cheng Chen. "Parallel Query on the In-Memory Database in a CUDA Platform." P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2015 10th International Conference on. IEEE, 2015.

[35] Kraft, Stephan, et al. "Wiq: work-intensive query scheduling for in-memory database systems." Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on. IEEE, 2012.

[36] Molka, Karsten, and Giuliano Casale. "Contention-Aware Workload Placement for In-Memory Databases in Cloud Environments." ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS) 2.1 (2016): 1.

[37] Muller, Stephan, et al. "Workload-aware aggregate maintenance in columnar in-memory databases." Big Data, 2013 IEEE International Conference on. IEEE, 2013.

[38] Yoon, Min, et al. "Conflict Prediction-Based Transaction Execution for Transactional Memory in Multi-core In-memory Databases." Cluster Computing (CLUSTER), 2016 IEEE International Conference on. IEEE, 2016.

[39] “Redis Benchmark Page”, http://redis.io/topics/benchmarks June 2015,

[40] Meyer, Robert, et al. "Assessing the Suitability of In-Memory Databases in an Enterprise Context." Enterprise Systems (ES), 2015 International Conference on. IEEE, 2015.

[41] Graefe, Goetz, et al. "In-memory performance for big data." Proceedings of the VLDB Endowment 8.1 (2014): 37-48.

[42] Szpisják, Patrik, and Levente Rádai. "Performance issues of In-Memory Databases in OLTP systems." Applied Computational Intelligence and Informatics (SACI), 2016 IEEE 11th International Symposium on. IEEE, 2016.

[43] Molka, Karsten, and Giuliano Casale. "Efficient Memory Occupancy Models for In-memory Databases." Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on. IEEE, 2016.

[44] Verma, Sudhir, and Vidur Shailendra Bhatnagar. "In-Memory Database Optimization Using Statistical Estimation." Cloud Computing in Emerging Markets (CCEM), 2015 IEEE International Conference on. IEEE, 2015.

[45] Schaffner, Jan, et al. "Predicting in-memory database performance for automating cluster management tasks." Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011.

[46] Mukherjee, Niloy, et al. "How does Oracle Database In-Memory scale out?." Software Technologies (ICSOFT), 2015 10th International Joint Conference on. Vol. 1. IEEE, 2015.

[47] Iwazume, Michiaki, et al. "Big Data in Memory: Benchmarking in Memory Database Using the Distributed Key-Value Store for Constructing a Large Scale Information Infrastructure." Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International. IEEE, 2014.

[48] Lee, Juchang, et al. "SAP HANA distributed in-memory database system: Transaction, session, and metadata management." Data Engineering (ICDE), 2013 IEEE 29th International Conference on. IEEE, 2013.

[49] Giles, Ellis, Kshitij Doshi, and Peter Varman. "Persisting in-memory databases using SCM." Big Data (Big Data), 2016 IEEE International Conference on. IEEE, 2016.

[50] Xu, Xiaofeng, Li Xiong, and Vaidy Sunderam. "D-grid: An in-memory dual space grid index for moving object databases." Mobile Data Management (MDM), 2016 17th IEEE International Conference on. Vol. 1. IEEE, 2016.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Mehmed Taha ARAS

Doğum Tarihi ve Yeri : 10.07.1990 – Amasya

Yabancı Dili : İngilizce

E-posta : m.taha.aras@gmail.com

ÖĞRENİM DURUMU

Derece	**Alan**	**Okul/Üniversite**	**Mezuniyet Yılı**
Doktora	Bilgisayar Mühendisliği	Yıldız Teknik Üniversitesi	halen
Yüksek Lisans	Bilgisayar Mühendisliği	Yıldız Teknik Üniversitesi	2016
Lisans	Yazılım Mühendisliği	Bahçeşehir Üniversitesi	2013
Lise	Fen Bilimleri	Kütahya Anadolu Öğretmen Lisesi	2008

İŞ TECRÜBESİ

Yıl	**Firma/Kurum**	**Görevi**
2015-halen	Türk Hava Yolları	Yazılım Mühendisi

YAYINLARI

Bildiri

1. M. T. Aras, Y. E. Selçuk, “Metric and Rule Based Automated Detection of Antipatterns in Object-Oriented Software Systems”, 7th International Conference on Computer Science and Information Technology (CSIT), IEEE Computer Society, 2016.