Bir Reklam Aracısı Yazılımının Mimari Evrimi

Ömer Kırkağaçlıoğlu[1], Görkem Giray[1], Arif Ersin[1], Cem Çatıkkaş[1], Sena Koçer[1], Tolga Şeremet[1], Murat Osman Ünalır[2]

[1] Kokteyl A.Ş., İstanbul, Türkiye
omer.kirk@kokteyl.com, gorkem.giray@kokteyl.com,
arif.ersin@kokteyl.com, cem.catikkas@kokteyl.com,
sena.kocer@kokteyl.com, tolga.seremet@kokteyl.com
[2] Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, İzmir
murat.osman.unalir@ege.edu.tr

Özet. Mobil cihazların ve uygulamaların yaygınlaşmasıyla birlikte mobil reklamcılık sektörünün önemi de son yıllarda artmıştır. Mobil reklamcılık işinin merkezinde yazılım sistemleri bulunmaktadır. Performans gibi kalite özellikleri gereksinimleri zorlayıcı hale gelen mobil reklamcılık sektöründeki yazılım sistemleri için bu değişimlere paralel olarak çözümler üretilmesi gerekmektedir. Bu bildiride bir reklam aracısı yazılımının değişen işlevsel ve kalite özellikleri gereksinimleri doğrultusunda mimarisinin nasıl evrildiği anlatılmaktadır. Ortaya çıkan, performans başlığı altında sınıflandırılabilecek problemlere literatürde bulunan mimari tasarım desenleri ve tasarım taktikleri kullanılarak çözümler üretilmiştir.

Anahtar Kelimeler: Yazılım Mimarisi Evrimi, Yazılım Mimari Tasarımı, Kalite Özelliği, Mobil Reklamcılık.

Architectural Evolution of an Ad Mediation Software

Abstract. With the widespread adoption of mobile devices and applications, the importance of mobile advertising industry has also increased in recent years. Software systems are at the heart of mobile advertising business. For software systems in the mobile advertising industry, where quality requirements such as performance are becoming more demanding, solutions must be produced in parallel with these changes. This paper describes how the architecture of an ad mediation software evolves to meet the changing functional and quality attribute requirements. The problems that can be classified under the heading of performance have been solved by using architectural design patterns and design tactics found in the literature.

Keywords: Software Architecture Evolution, Software Architecture Design, Quality Attribute, Mobile Advertising.

1 Giriş

Gerek dünyada gerekse Türkiye'de son birkaç yılda mobil cihazların kullanımı oldukça yaygınlaşmıştır. Bu cihazların kullanıcılar tarafından benimsenmesinin temel nedenlerinden biri bu cihazlar üzerinde çalışan uygulamaların sayısının artması, çeşitlenmesi ve kullanıcıların birçok gereksinimini karşılamalarına yardımcı olmalarıdır. Mobil uygulamaların kullanımının artmasıyla da çevrimiçi reklamcılık sektöründe mobil kanallara ağırlık verilmeye başlanmıştır. Bu bağlamda mobil uygulamada reklam gösterimini destekleyen çok sayıda reklam ağı ortaya çıkmıştır. Bu reklam ağlarının temel amacı reklam verenler ile reklam yayınlayanlar (bu durumda mobil uygulamalar) arasında bir köprü kurmaktır. Reklam ağlarının sayısının artmasıyla birlikte de çok sayıda reklam ağı ile reklam yayınlayanlar arasında köprü kurma görevini reklam aracıları üstlenmeye başlamıştır.

Mobil çevrimiçi reklamcılık sektörü tamamen yazılım üzerinde yapılandırılmıştır. Çevrimiçi reklam ağları, reklam aracıları ve reklam yayıncıları yazılım sistemleri aracılığıyla bu sektör içindeki görevlerini yerine getirmektedir. Dolayısıyla yazılım mühendisliği kapsamındaki tüm temel etkinlikler mobil çevrimiçi reklam aracılığı yazılımlarında da gerçekleştirilmelidir.

Bu çalışmada Kokteyl şirketinin reklam aracılık yazılımının mimari evrimi yazılım mühendisliği literatüründeki çalışmalarla bağlantı kurularak anlatılmıştır.

İkinci bölüm Kokteyl şirketi, mobil çevrimiçi reklamcılık sektörü ve şirketin reklam aracısı yazılımı hakkında genel bir bilgi vermektedir. Üçüncü bölümde reklam aracısı yazılımının mimari evrimi ve bu evrim sürecinde öğrenilen dersler anlatılmaktadır. Dördüncü ve son bölümde elde edilen sonuçlar ve gelecek çalışmalar sunulmuştur.

2 Arka Plan

Bu bölümde reklam aracılığı yazılımını geliştiren ve işleten Kokteyl şirketi hakkında genel bilgi verilmiş, şirketin yer aldığı mobil reklam sektörü ve şirketin yazılım ürünleri tanıtılmıştır. Son alt bölümde ise yazılım mimarisi evrimi çalışma alanı hakkında kısaca bilgi verilmiştir.

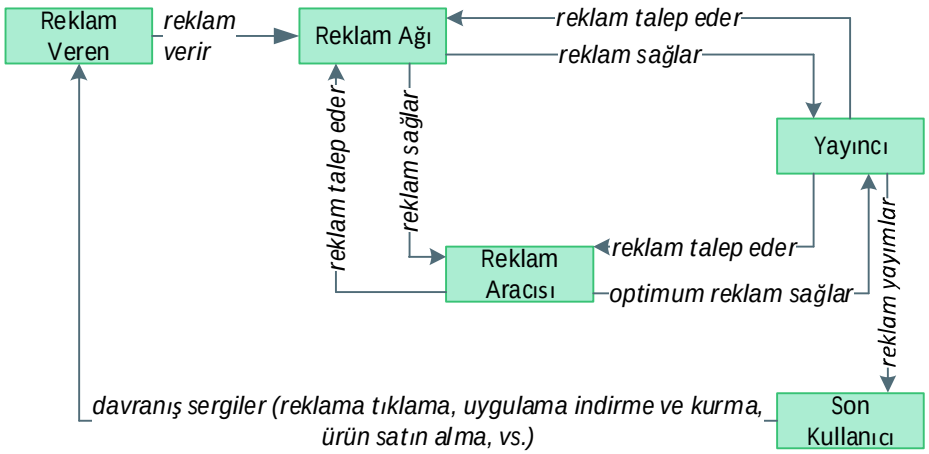
2.1 Şirket Hakkında Genel Bilgi

Kokteyl şirketi 2002 yılında web tabanlı uygulamalara odaklanmak üzere kurulmuştur. Sonrasında mobil uygulamaların ve bulut teknolojilerinin yaygınlaşmasıyla birlikte şirketin odağı web, mobil ve bulut uygulamalarına doğru evrilmiştir. Şirket çok sayıda yazılım ürününü geliştirmiş ve hizmete sunmuştur. Bu ürünlerden en önemlisi mackolik uygulamasıdır. Bu uygulama web ve mobil kanallar üzerinden spor etkinlikleriyle ilgili bilgiler ve istatistiksel veriler sunmaktadır. Bu uygulamanın bir kısmı 2012'de geri kalanı da 2016 yılında İngiltere'deki bir medya şirketi tarafından satın alınmıştır. 2008 yılından başlayarak şirket çevrimiçi reklamcılık sektörüne girmiş ve zaman içinde bir reklam aracısı yazılımı

geliştirmiştir. Şirket 2016 yılından bu yana bu reklam aracısı yazılımının geliştirilmesine ve işletilmesine yoğunlaşmıştır.

2.2 Mobil Reklam Sektörü

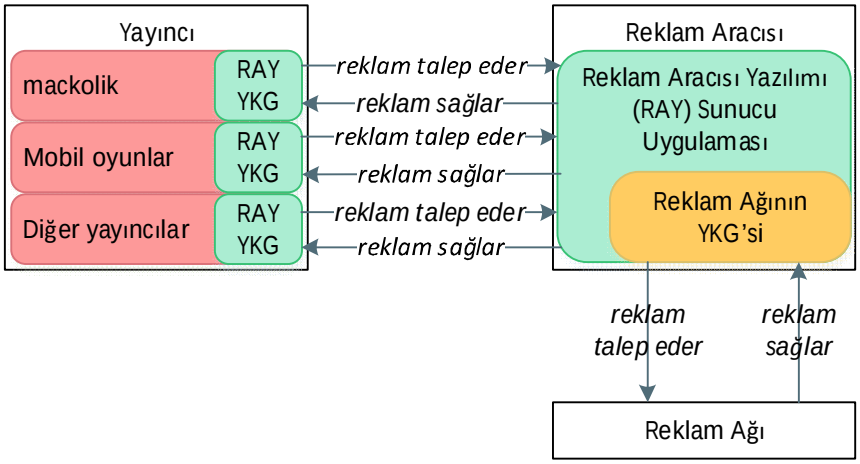
Çevrimiçi reklam sektöründeki ana paydaşlar Şekil 1'de gösterilmektedir. *Reklam verenler*, *reklam ağları*na ürünleri ve/veya hizmetleri hakkında tüketicilere bilgi vermek için reklam vermektedirler. *Yayıncılar*, *reklam ağları*ndan reklam talep etmekte ve *reklam ağları* yayıncıya mevcut durumdaki en uygun reklamı sağlamaktadır. Uygunluk konusundaki karar birçok değişkene (yayıncının hangi uygulama olduğu, son kullanıcı hakkındaki bilgiler, uygulamanın çalıştığı cihaz gibi) bağlı olarak değişmektedir. *Yayıncıları*n *reklam ağları*ndan doğrudan reklam alması durumunda bazı problemler ortaya çıkmaktadır. Teknik açından *yayıncı* uygulama reklam talep edeceği tüm *reklam ağları*yla entegre olmak zorundadır. İş amaçları açısından bakıldığında bir *yayıncı*nın birçok *reklam ağı*ndan optimum reklamı (en fazla geliri getirecek reklam) alamaması durumunda gelir kaybı yaşayacaktır. *Reklam aracıları* birçok *reklam ağı* ile entegre olarak optimum reklamı sağlama konusunda uzmanlaşmaktadır. Böylece *yayıncıları*n reklam gelirlerini arttırmasına yardımcı olmaktadır. Teknik açıdan ise çok sayıda *reklam ağı* ile entegre olmanın getirdiği karmaşıklık *reklam aracısı* tarafından yönetilmekte ve *yayıncıları*n sadece *reklam aracısı* ile entegre olmaları yeterli olmaktadır. *Son kullanıcılar* ise *yayıncı* uygulamaları kullanarak reklamları görüntülemekte ve bu reklamlara karşı zaman zaman bir davranış (reklama tıklama, uygulama indirme ve kurma, ürün satın alma gibi) sergilemektedir. Bu tür davranışlar *yayıncıla*ra gelir sağlamaktadır.



Şekil 1. Çevrimiçi reklamcılık sektöründeki ana paydaşlar.

2.3 Şirketin Yazılım Ürünleri

Şirketin yazılım ürünleri ve bu ürünlerin sektördeki diğer paydaşlarla olan ilişkileri Şekil 2'de gösterilmektedir. Şirketin reklam aracısı yazılımını oluşturan iki bileşen şekilde yeşil renk ile gösterilmiştir. Reklam Aracısı Yazılımının (RAY) sunucu uygulaması, reklam ağlarının yazılım geliştirme kitlerini (YKG) içermektedir. Bu YKG'ler aracılığıyla sunucu uygulaması reklam ağlarından reklam talep etmekte ve reklam almaktadır. Yayıncı uygulamalar ise RAY'nin YKG'sini uygulamalarının içine yerleştirerek RAY'den reklam talep etmekte ve almaktadırlar. Bir önceki bölümde anlatıldığı gibi, teknik açıdan RAY birçok reklam ağı ile entegre olmanın getirdiği karmaşıklığı yayıncı uygulamalara saydam hale getirmektedir. İş amaçları açısından ise RAY birçok reklam ağından sağladığı reklamlar arasından en uygun reklamı yayıncı uygulamaya göndererek reklam gelirlerinin arttırılmasını sağlamaktadır.



Şekil 2. Şirketin yazılım ürünleri.

Reklam Aracısı Uygulaması birçok işlevsel gereksinimi belirli kalite özelliklerini sağlayarak ve bazı kısıtlar altında karşılamaktadır. RAY'nin mevcut mimarisi zaman içinde birçok nedenden dolayı (değişiklik istekleri, yaşanan problemler gibi) evrilerek bugünkü halini almıştır.

3 Reklam Aracısı Yazılımının Mimari Evrimi

Bir yazılım sisteminin mimarisini belirleyen gereksinimler birçok biçimde karşımıza çıkmaktadır: metinler, modeller, mevcut sistemler, kullanım senaryoları, kullanım hikayeleri vs. Bu gereksinimler içinde mimari açıdan önemli gereksinimler mimari kararları etkilemektedir. Gereksinimler, biçimlerinden ve kaynaklarından bağımsız olarak üç ana sınıfta incelenebilmektedir [2]:

1. *İşlevsel gereksinimler:* Sistemin ne yapması gerektiğini, sistemin nasıl davranması gerektiğini ya da bir girdiye nasıl bir tepki vermesi gerektiğini tanımlar.
2. *Kalite özellikleri gereksinimleri (işlevsel olmayan gereksinimler):* İşlevsel gereksinimlerin ya da yazılım sisteminin niteliklerini belirtir. Bir işlevsel gereksinimin niteliği o işlevin ne kadar zamanda gerçekleştirilmesi gerektiğini ya da hatalı bir veri girişine karşı ne kadar dirençli olacağını belirtebilir. Bir yazılım sisteminin niteliği ise sistemin ne kadar zamanda canlı ortama konuşlandırılabileceğini ya da işletim maliyetleri için bir sınır olarak karşımıza çıkabilir.
3. *Kısıtlar:* Tasarım yaparken mutlaka uyulması gereken sınırlamalara işaret etmektedir. Örneğin, belli bir programlama dilini kullanma zorunluluğu, mevcut bir modülün kullanılma zorunluluğu gibi.

Yazılım mimarisi tasarımı ve gerçekleştirimi üzerinde derin etkileri olan ve yoklukları durumunda mimarinin çok farklı olabileceği gereksinimler mimari açıdan önemli gereksinimler olarak tanımlanmaktadır [2]. RAY'nin mimari tasarımını etkileyen gereksinimler aşağıdaki gibidir:

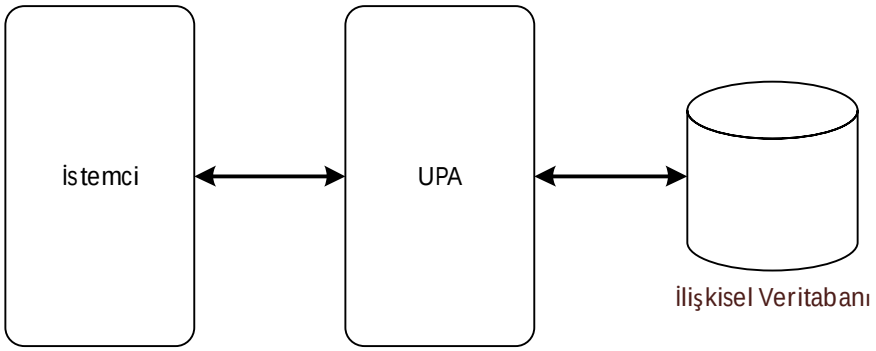
- G1. Aracılık ile ilgili gereksinimler: Hangi reklam ağından hangi sırada reklam isteneceğini belirleyen optimize edilmiş karar ağacının oluşturulmasını ve reklam gösterim istatistiklerini istemcilerden toplanmasını içeren gereksinimler.
    - G1.1. İstemcilerden gelen talepler doğrultusunda optimize edilmiş karar ağacının istemcilere gönderilmesi
    - G1.2. Reklam gösterim istatistiklerinin sunucu uygulamasında saklanması
- G2. Analitik gereksinimler: Reklam gösterim istatistiklerinin kullanıcı (mobil cihaz) bazında gruplanmasını ve kullanıcılar/cihazlar hakkındaki bilgilerin saklanmasını içeren gereksinimler.
    - G2.1. Kullanıcı/cihaz yaratılması ya da güncellenmesi
    - G2.2. Kullanıcının uygulama içinde yaptığı satın alma işlemlerinin izlenmesi
    - G2.3. Kullanıcıdan/cihazdan reklam isteklerinin gelmesi
- G3. Kampanya izleme gereksinimleri:
    - G3.1. Bir isteğin ilgili kaynağa yönlendirilmesi
    - G3.2. Bir istek yönlendirildikten sonra gerekli istatistiklerin güncellenmesi

Birçok yazılım sisteminin mimarisi, yeni istekler, yeni kalite özelliği gereksinimleri, teknolojideki değişimler ya da başka nedenlerle değişikliğe uğrar [1]. Nedeni ne olursa olsun yazılım sektöründeki projelerde mimarinin evrilmesi olağandır.

3.1 Aracılık Bileşeninin Mimarisinin Evrimi

Yekpare mimariye sahip bir yazılım sistemi tüm işlevleri yerine getiren tek bir bileşenden oluşur. Sistemde yapılacak her değişiklikte sistemin tamamının yeni

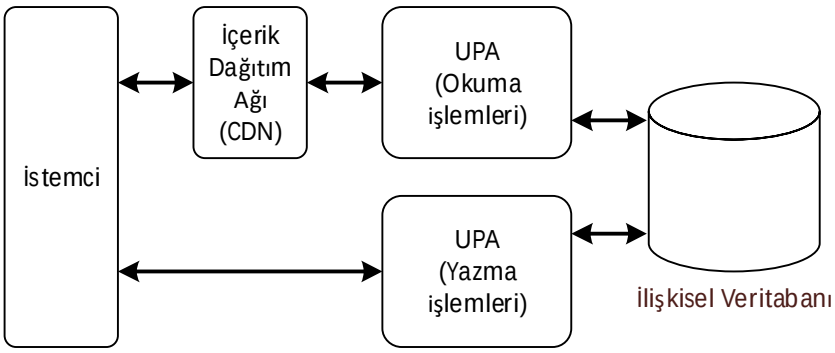
versiyonu konuşlandırılır. Şekil 3'te RAY'nin yekpare mimari yaklaşımıyla tasarlanmış ilk mimarisi gösterilmektedir. Bu mimaride tüm işlevler bir uygulama programlama arayüzü (UPA – API: Application Programming Interface) ile istemcilere sunulmuştur. Veri saklama için ilişkisel bir veritabanı (PostgreSQL) kullanılmıştır.



Şekil 3. Birinci aşamadaki yekpare mimari.

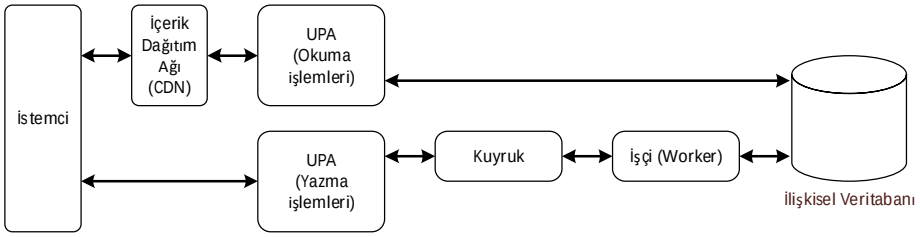
Yekpare mimarinin bir avantajı, değişikliklerin canlı sistemde devreye alınmasının görece kolay olmasıdır (bir UPA'nın konuşlandırılması). Bunun yanında yazılım sisteminin anlaşılmasının kolay olması da hataların bulunmasını da kolaylaştırmaktadır. Diğer taraftan istemcilerden gelen isteklerin artmasıyla ve belirli zaman aralıklarında yoğunlaşmasıyla birlikte yekpare mimarinin performans açısından problemleri ortaya çıkmaya başlamıştır. G1.1 ve G1.2 gereksinimlerinin gerektirdiği çözümler birbirinden farklı olmasına rağmen (ilkinde veritabanından okuma, ikincisinde veritabanına yazma işlemleri yoğun) yekpare mimari nedeniyle aynı mimari çözüm uygulanmak zorunda kalınmıştır. UPA'nın sık güncellenmesi gereken durumlarda hangi işlevde güncelleme yapılırsa yapılsın tüm UPA'nın belirli bir süre devre dışı kalması söz konusu olmaktadır. Bunun yanında zaman zaman reklam gösterim istatistiklerinin saklanması için veritabanına gelen yazma isteklerinin sunucuda oluşturduğu yoğunluk nedeniyle karar ağacı gönderim taleplerine de cevap verilememe durumu ortaya çıkmıştır.

Performans problemlerini çözmek için ilgilerin ayrılığı (separation of concerns) ilkesi [5] doğrultusunda veritabanından okuma ve veritabanına yazma işlemleri ayrı UPA ile yapılmaya başlanmıştır. Okuma işlemlerinde performansı arttırmak için içerik dağıtım ağı (İÇA – CDN: Content Delivery Network) kullanılmaya başlanmıştır [3]. İÇA kullanımı ile birlikte istemci taleplerine dönüş performansı artmış ve okuma işlemlerinin erişilebilir olma yüzdesinde, literatürdeki bulgularla paralel olarak [4], yükselme gözlemlenmiştir. Ayrıca hem okuma işlemleri hem de yazma işlemleri için gelen istekler yük dengeleyiciler (load balancer) ile farklı sunuculara dağıtılmaya başlanmıştır. Şekil 4'te yekpare bileşenin iki sorumluluğa sahip iki ayrı bileşen haline getirildiği mimari gösterilmektedir.



Şekil 4. İkinci aşamadaki iki ana bileşenli mimari.

Okuma ve yazma sorumluluklarının ayrı bileşenlere verilmesinden sonra yazma işlemlerinde sistemin yoğun kullanıldığı zamanlarda performans problemleri yaşanmaya başlandığı görülmüştür. Her yazma talebi veritabanı yönetim sistemi tarafından karşılandıktan sonra istemcilere cevap dönülmesi, tepki süresini veritabanı sunucusunun performansına bağlı kılmıştır. Dolayısıyla veritabanı sunucusundaki problemlerde ya da yavaşlıklarda istemcilerin taleplerine tepki süresi artmaya başlamıştır. Veritabanında saklanan verinin her an tutarlı olma gereksinimi olmadığı için gelen yazma isteklerinin bir kuyrukta biriktirilmesi ve işçiler tarafından belirli aralıklarla veritabanına yazılması tercih edilmiştir. Web-Kuyruk-İşçi mimari stiline [3] karşılık gelen bu mimari tasarım sonucunda sistemin mimarisi Şekil 5'te gösterilmektedir.



Şekil 5. Üçüncü aşamadaki kuyruk ve işçi bileşenleri eklenmiş şekliyle mimari.

Bu mimarinin gerçekleştirimi için koşut zamanlı programlamayı destekleyen Golang [6] dili kullanılmıştır. Yazma işlemlerinden sorumlu UPA gelen talebi kuyruğa ekler ve istemciye cevap dönmektedir. Bir veri birleştirici kuyruğu kontrol eder; veriyi biriktirir ve belirli bir sürede ya da veri belirli bir hacme ulaştığı zaman veriyi işçiye gönderir. İşçi birikmiş veriyi kuyruktan alır; geçici bir veritabanı tablosuna bu veriyi yazar; geçici tablodaki veriyi kalıcı tabloya ekler ve geçici tabloyu temizler. İşçi, veri transferi ve ortak tablolara veri yazma işiyle uğraştığı için olası bir kilitlenmeyi (deadlock) engellemek için tek bir işçi kullanılmaktadır. Bu mimarinin olumsuz tarafı ise işçinin kuyruğa eklenen veriyi yeterince hızlı biçimde kalıcı

tablolara taşıyamaması durumunda biriken verinin bellek ve işlemci kaynaklarında darboğaz oluşturması durumuyla sonuçlanmaktadır. Bu problemi gidermek için ise veriyi kuyruktan geçici tablolara aktarma işi ve geçici tablodan kalıcı tabloya veri yazma işi farklı süreçlere paylaştırılmıştır. Bu tasarımda veriyi kuyruktan geçici tablolara aktarma işini birden fazla süreç üstlenebilmektedir. Geçici tablodan kalıcı tabloya veri aktarımı ise yine bir süreç ile sağlanmaktadır. Mimarinin bu son aşamadaki haliyle reklam gösterim isteklerinin sorunsuz biçimde karşılanması sağlanmaktadır.

3.2 Analitik Bileşeninin Mimarisinin Evrimi

Analitik bileşeninin temel görevi kullanıcı bazlı verinin istemcilerden toplanması ve saklanmasıdır. Bu temel görev yeni kullanıcıların kayıtlarının yaratılması (G2.1), gerektiğinde bilgilerinin güncellenmesi (işletim sistemi, ülke, vs gibi) (G2.2), yayıncı uygulamada yaptığı satın alma verisinin toplanması ve uygulamada gösterilen reklamların verisinin toplanmasını kapsamaktadır. Aracılık bileşeninde reklam gösterim verisi reklam ağları bazında saklanırken analitik bileşende bu verilerin kullanıcı bazında saklanması gerekmektedir. Bu gereksinim, mimarinin yoğun bir veri yazma işiyle başa çıkabilecek şekilde tasarlanmasını ve gerçekleştirilmesini gerektirmektedir. Bu gereksinimi karşılayabilmek için NoSQL bir veritabanı [7] kullanılmıştır.

Bu bileşenin yazma görevini yerine getirmesi için kuyruk ve işçi bileşenleri kullanılmıştır. Gelen istekler bir kuyrukta biriktirilmekte ve toplu halde bir NoSQL veritabanı olan MongoDB'ye [14] yazılmaktadır. Bu çözüm yeni kullanıcı kaydı yaratılması, kullanıcı bilgilerinin güncellenmesi gibi görevler için gereksinimi karşılamıştır. Ancak yayıncı uygulamada yapılan satın alma işlemlerinin doğrulanması görevinin yerine getirilmesi noktasında bu mimaride problemler ortaya çıkmıştır.

Bir satın alma işleminin, bir kullanıcı adına veritabanına kaydedilmeden önce bu satın alma işleminin sahte olup olmadığının kontrol edilmesi gerekmektedir. Bu kontrolü yapmak için Apple'ın ve Google'ın sunduğu servislerden faydalanılmaktadır. Ancak sistemin genel performansı bu servislerin performansına doğrudan bağlı hale gelmiştir. Bazı durumlarda bu servislerdeki yavaşlıktan dolayı sistemin genel performansında düşüş gözlemlenmiştir. Bunu engellemek için satın alma işleminin kontrolünü gerçekleştiren işlev ayrı bir bileşen haline getirilmiştir. Böylece istemciden alınan veriler geçici olarak saklanmakta, istemciye cevap dönülmekte ve başka bir bileşen tarafından satın alma işleminin kontrolü yapılmaktadır. Bu kontrolden cevap geldikten sonra hem ilişkisel hem de NoSQL veritabanında gerekli alanlar güncellenmektedir. Senkron olarak yapılan işlerin asenkron hale getirilmesi performans artışına neden olmuştur.

Analitik bileşeninin karşıladığı bir başka gereksinim yayıncı uygulama sahiplerine uygulamalarını belirli bir süre içerisinde aktif olarak kullanan tekil kullanıcı sayısını vermektir. Mobil reklamcılık sektöründe de bu sayılar genellikle 1, 7, 14 ve 30 gün için sağlanmaktadır. Bu sayıları sağlamak için gerekli veriyi saklayan MongoDB'deki bilgi sorgularken karşılaşılan zorluklardan dolayı son 30 günlük tüm verinin ilişkisel

bir veritabanına yazılmasına karar verilmiştir. Bu da kullanıcı bazlı verinin hem MongoDB'ye hem de ilişkisel veritabanına yazılma gerekliliğini doğurdu. Ancak ilişkisel veritabanına yazma sürelerinin uzunluğu nedeniyle yoğun istek gelen zamanlarda sistem performans gereksinimlerini karşılayamadı.

Bu problemi gidermek için hem hızlı yazma imkanı veren hem de verinin raporlanması için gerekli yetenekleri sağlayan Redis veritabanı [8] sisteme eklenmiştir. Redis veritabanı kullanıcı bazlı verileri gün içinde saklamakta gün sonunda ise bu veriler ilişkisel veritabanına aktarılmaktadır. Bu değişiklik ile birlikte hem kullanıcı bazlı verinin saklanması hem de istendiğinde raporlanması gereksinimleri karşılanmıştır.

3.3 Kampanya İzleme Bileşeninin Mimarisi

Bu bileşen, kritik müşterilerin (yayıncı uygulamalar) RAY'den aldığı linkler ile kendi uygulamalarının reklamlarından aldıkları tıklamaları takip ederek hangi kaynaktan ne tür kullanıcı aldıklarını analiz etmelerini sağlamaktadır. Reklam verenin para ödeyerek verdiği reklamlara tıklandığında öncelikle bu bileşenin sunduğu servis çağrılır. Dolayısıyla bu bileşenin yüksek oranda kullanılabilir olması gerekmektedir. Bu servis tetiklendikten sonra takip için gerekli bilgiler toplanır ve Google veya Apple uygulama dükkanlarındaki uygulama sayfasına yönlendirme yapılır. Servisin çalışır durumda olmaması durumunda bu yönlendirme yapılamaz ve reklam verenin kullanıcı kazanmak için yaptığı reklam boşa gitmiş olur. Bu önemli gereksinimi karşılayan servis Google Cloud'da barındırılmaktadır. Bu servise gelen isteklerin yerine getirilmesi için ilgili verinin sürekli tutarlı olması gerekmektedir. Bundan dolayı bir isteğe gerekli tüm adımlardan geçildikten sonra cevap dönülmektedir.

4 Sonuçlar ve Gelecek Çalışmalar

Yekpare bir mimari tasarım ile başlayan RAY sunucu uygulaması, üç farklı ana bileşen ve üç yan süreç olarak bölünmüştür. Her bileşende gereksinimlerin farklılıkları ve servislerin bu farklılıklarına uygun tasarım yapma fikri bulunmaktadır. Trafik yoğunluğu arttıkça ana bileşenlerin daha küçük parçalara ayrılma ihtiyacı gündeme gelecektir. Bu da aslında yekpare mimariden mikroservis mimarisine doğru bir evrim [13] anlamına gelmektedir. Mikroservislerin içinde barındırdığı takip etme (loglama), performans ölçümü, bir isteği farklı servisler arasında izleyebilme, hata ayıklama ve yayımlama konusundaki zorluklarını çözmek için birtakım araçlar gerekmektedir. Mevcut mimaride log dosyalarını farklı sunuculardan ortak bir yerde toplayıp işlemek bir sorun haline gelmiştir. Önümüzdeki dönemde trafik daha da arttıkça performansı izlemek ve log dosyalarını analiz ederek sorun olan yerleri bulabilmek daha kritik hale gelecektir. Bu sorunları çözmek için ELK (Elastic Search, Logstash, Kibana) altyapısını [12] sistemimize entegre etmeyi planlanmaktadır.

Mimarinin evrim sürecinde ilgilerin ayrılığı ilkesiyle ve tutarlılık gereksinimlerine göre yapılan, işlemlerin istemci isteklerinin yoğunluğundan bağımsız hale getirilmesi Golang dili kullanılarak yapılmıştır. Her ne kadar bu bileşenler yüksek performansla

çalışabilseler de Golang'in özelliklerinden kaynaklanan bazı güvenilirlik problemleri bulunmaktadır. Örneğin Golang'de bir kuyruktan okunan bir mesaj okunduğu anda silinmekte ve mesajın işlenmesi sırasında bir sorun oluştuğunda bu mesaja tekrar erişilememektedir. Benzer şekilde çalışma zamanında zaman aşımı hatası olduğunda o hatayı işleyerek bir mesaj göndermek, o işi kolayca durdurmak gibi özellikler mevcut değildir. Kuyruk-işçi mimarisi içeren kritik bileşenlerin gelecekte Erlang ile geliştirilmesi planlanmaktadır. Erlang'da bulunan OTP kütüphanesi bu problemler için çözüm içermesinin yanında yüksek oranda kullanılabilir durumda olma ve eş zamanlı işlem yapabilme konularında da çeşitli avantajlar sağlamaktadır [9][10].

Daha güvenilir kalıcı kuyruklar için de Kafka servisinin entegre edilmesi planlanmaktadır. Bir publish/subscribe servisi olan Kafka [11], belli sınıftaki işleri takip eden işçiler ve bu sınıflara yayın yapan dağıtıcı mimarisini, kuyruktaki mesajlar üzerinde Golang'de olmayan birçok özellik ekleyerek sağlamaktadır. Bunun yanında Kafka'ya yazılan mesajlar, servisin yeniden başlatılması veya çökmesi sırasında diske yazılarak veri kaybı önlenmiş olmaktadır. Golang'in kuyrukları bellekte saklandığı için servislerin istem dışında yeniden başlatılması gibi durumlarda veri kaybı yaşanabilmektedir.

Kaynakça

1. Barnes, J.M., Garlan, D. & Schmerl, B. Softw Syst Model (2014) 13: 649. https://doi.org/10.1007/s10270-012-0301-9
2. Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (3rd ed.). Addison-Wesley Professional.
3. Cloud Application Architecture Guide, Mike Wasson, Masashi Narumoto and the Microsoft Patterns and Practices team, 2017, Microsoft Corporation.
4. Balachander Krishnamurthy, Craig Wills, and Yin Zhang. 2001. On the use and performance of content distribution networks. In Proceedings of the 1st ACM SIGCOMM Workshop on Internet measurement (IMW '01). ACM, New York, NY, USA, 169-182. DOI: https://doi.org/10.1145/505202.505224
5. Hürsch, Walter L., and Cristina Videira Lopes. Separation of concerns. (1995).
6. https://golang.org/ref/spec Erişim tarihi: 18 Haziran 2018.
7. Rick Cattell. 2011. Scalable SQL and NoSQL data stores. SIGMOD Rec. 39, 4 (May 2011), 12-27. DOI: https://doi.org/10.1145/1978915.1978919
8. Jing Han, Haihong E, Guan Le and Jian Du, "Survey on NoSQL database," 2011 6th International Conference on Pervasive Computing and Applications, Port Elizabeth, 2011, pp. 363-366. doi: 10.1109/ICPCA.2011.6106531
9. S. Vinoski, "Concurrency with Erlang," in IEEE Internet Computing, vol. 11, no. 5, pp. 90-93, Sept.-Oct. 2007. doi: 10.1109/MIC.2007.104
10. Jim Larson. 2009. Erlang for concurrent programming. Commun. ACM 52, 3 (March 2009), 48-56. DOI: https://doi.org/10.1145/1467247.1467263
11. Kreps, Jay, Neha Narkhede, and Jun Rao. "Kafka: A distributed messaging system for log processing." Proceedings of the NetDB. 2011.
12. J. Bai, "Feasibility analysis of big log data real time search based on Hbase and ElasticSearch," 2013 Ninth International Conference on Natural Computation (ICNC), Shenyang, 2013, pp. 1166-1170. doi: 10.1109/ICNC.2013.6818154

13. Villamizar, Mario, et al. "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud." Computing Colombian Conference (10CCC), 2015 10th. IEEE, 2015.
14. Chodorow, Kristina. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage. " O'Reilly Media, Inc.", 2013.