Mobil Uygulamalarda Vekil Tabanlı Kod Taşıma Yönteminin Farklı Seviyelerdeki Bulut Bilişim Altyapılarının Kullanılması Durumundaki Başarımının Karşılaştırılması

Mahir Kaya[1], Altan Koçyiğit[1]

[1]Enformatik Enstitüsü, Orta Doğu Teknik Üniversitesi, Ankara, Türkiye

{mkaya,kocyigit }@ metu.edu.tr

Özet. Günümüzde akıllı telefonların kullanımı gün geçtikçe artmaktadır. Akıllı telefon ve kablosuz iletişim teknolojilerindeki gelişmeler sayesinde insanlar bilgiye artık her yerde ve her zaman erişebiliyorlar. Akıllı telefon teknolojisi ilerlemesine rağmen yine de hafıza ve işlemci gücü kapasitesi bakımından masaüstü veya sunucu bilgisayarlara göre çok geride kalmaktadır. Bunun yanında akıllı telefonların enerjisinin çabuk tükenmesi başlı başına önemli bir problem olmaktadır. Akıllı telefonların bu olumsuzlukları yakındaki veya uzaktaki bulut bilişim teknolojileri ile iyileştirilebilmektedir. Özellikle resim işleme, nesne tanıma ve artırılmış gerçeklik gibi işlem ve hafıza yoğunluklu uygulamaları çalıştırabilmek daha kolay olabilecektir. Bu çalışmada akıllı telefonlardaki hesaplama yoğunluklu kod parçacıklarını uzaktaki bir buluta taşıma yöntemleri olan Uzak Metot Çağırma [1], Android Arayüz Tanımlama Dili [2] kullanılarak oluşturulan servisler ve OSGi servisleri [3] ile bu işlemin yapılması çalışmalarını inceledik. Vekil tabanlı kod taşıma yöntemini kullanarak nesne tanıma uygulaması için hesaplama yoğunluklu işleri buluta yaptırdık. Nesne tanıma uygulaması için %60 ile %83 oranında işlem süresi ve %65 ile %88 oranında enerji tasarrufu sağladık.

Anahtar Kelimeler: mobil hesaplama, bulut bilişim, AIDL, OSGi, uzak metot çağırma

1 Giriş

Bilgi teknolojilerinin gelişmesine paralel olarak mobil cihazların (akıllı telefon ve tablet) kullanımı gün geçtikçe artmaktadır. Mobil iletişim teknolojilerindeki (WiFi, 3G) gelişmeler ile kullanıcılar her yerden anlık bilgilere kolayca erişebilmektedirler. Uluslararası İletişim Birliği (ICT) gelişim raporlarına göre mobil ve sabit geniş bant üyeliği dünya genelinde artmaktadır [4]. Buna ilaveten, Uluslararası Telekomünikasyon Birliğinin (ITU) tahminlerine göre internet kullanım oranı 2013 yılı sonunda %40'a ulaşmıştır [4]. Hızla gelişen bu teknolojiler zaman ve maliyet açısından kullanıcılara yönelik birçok faydalar sunmaktadır. Eskiden e-posta

görüntüleme, gönderme, fotoğraf çekme, görüntüleme ve gönderme, anlık mesajlaşma gibi nispeten basit uygulamalar için yaygın olarak kullanılan akıllı telefonlar, günümüzde daha karmaşık uygulamalar için de kullanılmaktadır. Öte yandan, akıllı telefonlar ve tabletler sınırlı kapasiteye sahip olmalıdır çünkü kullanıcılar kullanım kolaylığı sağlayan büyüklük ve incelik kriterlerine önem vermektedir [5]. Bu yüzden, bu mobil cihazların hafıza, depolama ve işlemci gücü açısından masaüstü ve sunucu bilgisayar kapasitelerine ulaşması şimdilik beklenmemektedir. Mobil cihazların en büyük sorunlarından biri de çalıştırılan uygulamalara göre enerjilerinin çabuk tükenmesidir. Özellikle yoğun hesaplama gerektiren uygulamalarda enerji tüketimi büyük bir sorun teşkil etmektedir. Bunun yanında GPS, kamera, yüksek-çözünürlüklü ekran ve yüksek hızlı kablosuz internet arayüzleri gibi birçok özellik de akıllı telefonlarda enerji tüketimini ciddi miktarda artırmaktadır.

Akıllı telefonlar hesaplama yoğunluklu işlerini yakındaki veya uzaktaki bir bulut bilişim altyapısına taşıyarak yukarda söz edilen özelliklerini destekleyebilirler. Akıllı telefonların bulut bilişim altyapısından yararlanmaları için genelde iki yöntem üzerinde yoğunlaşılmıştır. Birinci yöntem, sanal makinenin (VM) tamamen buluta taşınıp tekrardan çalıştırılıp hazır hale getirilerek hesaplamaların yapılıp geri dönülmesidir. Bu yöntemde ağ maliyeti çok fazla olmakla birlikte hesaplama anında telefon algılayıcıları kullanılmak istenildiğinde problemler yaşanmaktadır. İkinci yöntem, kod taşıma yöntemidir. Bu yöntemi üç alt başlık altında toplayabiliriz. Bunlardan birincisi Uzak Metot Çağırma [1] ile vekil (proxy) tabanlı yöntem, ikincisi Arayüz Tanımlama Dilini [2] kullanarak servis hazırlama yöntemi ve son olarak da OSGi servis [3] tabanlı yöntemdir. Bu çalışmada vekil tabanlı yöntemi kullanarak buluta kod taşıma ile hesaplama yoğunluklu işlemleri cihazın erişebildiği bir sunucuda yapıp sonucu telefona geri döndürdük. Deneysel çalışmalarımızı nesne tanıma uygulaması ile yaptık. Bu uygulamayı telefonda, yerel alandaki bir buluta WiFi teknolojisi üzerinden erişim sağlayarak ve uzaktaki bir buluta WiFi ve 3G iletişim teknolojisi ile erişim sağlayarak çalıştırıp elde edilen başarımları karşılaştırdık. Yerel alandaki bulutta çalıştırma, telefonda çalıştırmaya göre enerji tüketiminde %65 ile %88 oranında ve işlem sürelerinde %60 ile %83 oranında tasarruf sağlarken, özellikle 3G ve uzaktaki bir bulut kullanıldığı zaman başarım olumsuz etkilenmekte, yani buluta taşıma daha yüksek maliyetli olabilmektedir.

Bu bildiri şu şekilde organize edilmiştir: Bölüm 2'de kod taşıma yöntemleri ile ilgili literatürdeki çalışmalara yer verilmiştir. Bölüm 3'de mobil hesaplama için kod taşıma yöntemleri açıklanmıştır. Bölüm 4'de uyguladığımız kod taşıma yöntemi verilmiştir. Bölüm 5'de örnek uygulama açıklanıp performans değerlendirme sonuçları verilmiştir. Son olarak Bölüm 6'da çalışma sonuçları özetlenmiştir.

2 İlgili Çalışmalar

Chun ve arkadaşları [6] ve Satyanarayanan ve arkadaşları [7] sanal makine taşıma yöntemini uygulayarak akıllı telefonların sanal makinesini buluta taşıdıktan sonra çalıştırıp hesaplama yoğunluklu işleri yaptırmışlardır. Akıllı telefonların sanal makinesini buluta taşımak yüksek ağ maliyeti getirmesinin yanında hesaplama

zamanında telefonun algılayıcılarının kullanılmasının gerektiği durumlarda geri dönüşler kolay bir şekilde çözülememektedir. Chun ve arkadaşları [6] çalışmasında öncelikle akıllı telefonun sanal makinası daha sonra hesaplama yoğunluklu iş parçacıkları buluta taşınmaktadır.

Verbelen ve arkadaşları [8,9] Android uygulamaları için OSGi servis yapısının kullanılması için bir çerçeve yazılımı geliştirmiştir. Android uygulama geliştirme modeli OSGi modül sisteminden farklı olduğu için OSGi servis geliştirme ve servislerin bulunması için Apache Felix gibi özel yazılım araçlarının uygulama geliştirilecek olan her akıllı telefonda yerleştirilmesi gerekmektedir. Ayrıca, Android uygulamalarının OSGi paketleri şeklinde yerleştirilebilmesi için hesaplama yoğunluklu iş parçacıklarının uygun OSGi paketlerine dönüştürülmesi gerekmektedir. Bu çalışmada, Eclipse uygulama geliştirme ortamı için dönüştürme işlemini yapan bir yazılım aracı geliştirmişlerdir. Ayrıca, uygulama geliştiricilerin hesaplama yoğunluklu iş parçacıklarını işaretlemesi de gerekmektedir.

Kemp ve arkadaşları [10] Cuckoo adında bir çerçeve yazılımı geliştirmişlerdir. Bu yazılımda Android Arayüz Tanımlama Dili (AIDL) kullanılarak hesaplama yoğunluklu iş parçacıkları, Android servislerine dönüştürülüp buluta taşınmıştır. Bu çalışmada geliştirilen servisler akıllı telefonda çalışıyor olsa bile bu servislerle iletişim, Süreçler Arası İletişim (IPC) yöntemi ile gerçekleştirildiğinden uygulamaya ek bir iletişim yükü getirmektedir. Cuckoo yazılımı, uygulama geliştirici tarafından AIDL kullanılarak yerel servisler şeklinde yazılan iş parçacıklarını uzak makinada çalışacak servislere dönüştürmektedir. Bulutta çalışacak servislerin akıllı telefon kaynaklarına veya paylaşılmış hafıza alanlarına erişmesi gerektiği durumlar için çözümler sunulmamıştır. Kovachev ve Klamma [11] Cuckoo çerçeve yazılımına Tamsayılı Doğrusal Programlama yöntemi ile iş parçacıklarının buluta taşınması karar modelini eklemişlerdir. Bu karar modeli ile ortamdaki ağ veri aktarımı ve iş parçacıklarının işlem süresine göre uygun olanların buluta taşınmasına karar verilmektedir. Chen ve arkadaşları [12] AIDL servis geliştirme yöntemini kullanarak Android telefonlar için kod taşıma yöntemi geliştirmiştir. Zhang ve arkadaşları [13] bayt kod enstrümentasyonu yöntemi ile çalışma zamanında buluta taşınacak iş parçacıklarını belirlemeye çalışmışlardır. Bu yöntemde uygulamalar imzalandıktan sonra bayt kod enstrümentasyonu yapıp uygulamayı değiştirmek sorunlar yaratmaktadır. Cuervo ve arkadaşları [14] ve Kristensen ve Bouvin [15] Uzak Metot Çağırma yöntemini kullanmıştır. Bu çalışmalarda da hesaplama yoğunluklu iş parçacıklarının uygulama geliştirici tarafından işaretlenmesi gerekiyor.

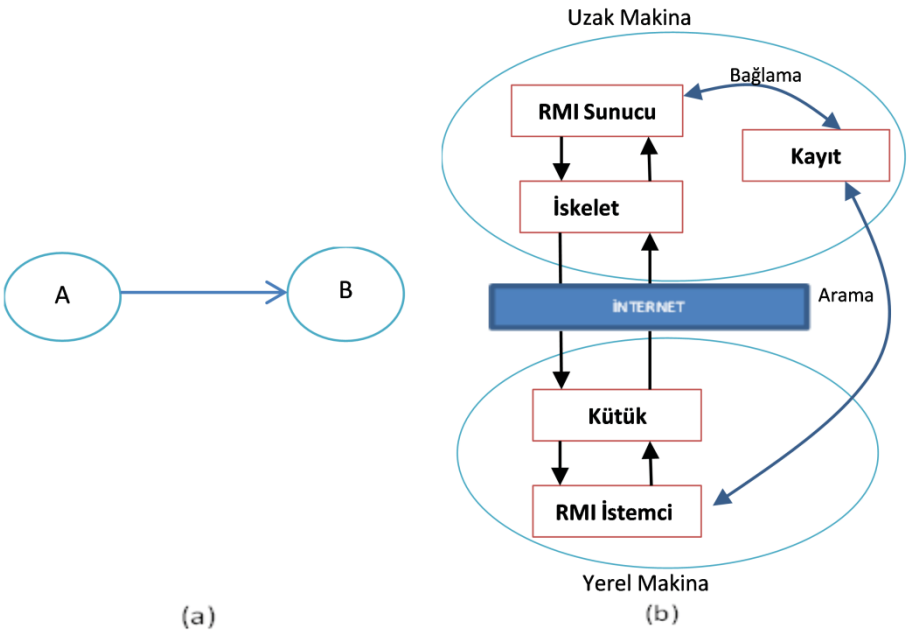
3 Mobil Hesaplama için Kod Taşıma Metotları

Akıllı telefonlarda hesaplama yoğunluklu işlemleri buluta yaptırmak için genel olarak iki yöntem kullanılmaktadır. Bunlardan birinci yöntem akıllı telefondaki sanal makinenin tamamen buluta taşınıp, orda çalıştırıp sonucun geri dönülmesidir, fakat bu yöntem bildirinin kapsamı dışında tutulduğundan anlatılmayacaktır. İkinci yöntemi ise üç alt başlık altında inceleyeceğiz ve bunlar sırasıyla vekil tabanlı sınıflar ile Uzak Metot Çağırma yöntemi, Arayüz Tanımlama Dillerini kullanarak servisler oluşturup

bu servislerin kullanılması yöntemi ve son olarak da OSGi servis yöntemi açıklanacaktır.

3.1 Uzak Metot Çağırma Yöntemi

Sanal makineler (Java Sanal Makinesi-JVM) bir nesnenin referansı vasıtasıyla ilgili nesnenin istenilen metodunu çağırma görevini yerine getirirler. Şekil 1(a)'da olduğu gibi A sınıfı, B sınıfının yerel referansını sanal makineden alarak B sınıfının metotlarını çağırabilmektedir. Eğer B sınıfı farklı bir sanal makinede ise bu şekilde B sınıfının metotlarının çağrılması mümkün olmamaktadır çünkü A sınıfı B sınıfının yerel referansını tutmaktadır. Bu yüzden farklı bir makinedeki bir nesnenin metotlarını açık bir şekilde çağırmak istiyorsak Uzak Metot Çağırma (RMI) gibi mekanizmalar kullanmak zorundayız.



Şekil. 1. Metot Çağırma Yöntemleri (a) Yerel Metot Çağırma (b) Uzak Metot Çağırma

Uzak Metot Çağırma yöntemi ile dağıtık sistemlerdeki klasik Uzak Fonksiyon Çağırma (RPC) mekanizmasına benzer bir şekilde uzaktaki bir bilgisayar üzerinde çalışan nesnenin metotlarını çağırabilmek mümkün olmaktadır. Bu yöntemde şekil 1(a)'daki B sınıfı uzak bir bilgisayarda çalışıyor olmasına rağmen A sınıfı B'ye yerel bir makinada çalışıyor gibi metotlarına erişebilir ve kullanabilir. İstemciler uzak servisleri bulabilmek için bir isimlendirme ve dizin servisini kullanılırlar. Sunucu tarafında istemciler tarafından kullanılacak olan nesneler bu servise-JNDI (Java Naming and Directory Interface) bağlanır ve şekil 1(b)'de görüldüğü gibi istemciler

bu servisin arama (“lookup()”) metodunu kullanarak istedikleri “uzak nesne” in referansını elde ederek metotlarını çağırabilmektedirler.

3.2 Arayüz Tanımlama Dili (IDL) Yöntemi

Akıllı telefonlardaki hesaplama yoğunluklu işleri uzaktaki bir sunucuya, Android işletim sisteminin sunduğu IDL'i kullanarak da gerçekleştirebiliriz. Android işletim sisteminde en önemli uygulama bileşenleri aktiviteler ve servislerdir. Aktiviteler bu işletim sisteminin grafik arayüzleridir. Servisler ise hesaplama yoğunluklu işleri uygulamanın arka tarafında gerçekleştirebilen bileşenleridir. Android servis mekanizması grafik arayüz ile uygulama mantığını birbirinden ayırmaktadır. Bu sayede grafik arayüzdeki işlemlere bir müdahalede bulunmadan hesaplama yoğunluklu işleri arka tarafta ayrı bir iş parçacığında gerçekleştirmektedir. Bir Android servis ilk çalıştırıldığında aktiviteleri yerleştirmektedir ve bu aktiviteler ile istenilen servise bağlantı yapılarak bu servisler çalıştırılabilmektedir. Android servisler, süreçler arası iletişim (IPC) mekanizması ile bağlanılırlar. Bir servis birden çok aktivite ile bağlantı kurabilir. Android IPC mekanizması önceden aidl dosyasında tanımlanan servisleri ön derleme aşamasında bir kütük/vekil (stub/proxy) ikilisi yaratarak kullanıma sunmaktadır.

Android Arayüz Tanımlama Dili olarak adlandırılan AIDL hem servis sağlayıcı hem de servisi kullanan tarafından kullanılmak zorundadır. Servis metotlarını çağırmak için kullanılan vekil, parametre olarak ilkel tipleri ve Java programlama dilindeki bit dizisine çevrime (serialization) benzer parsel sınıfını uygulamak zorunda olan nesneler kullanılabilir.

Kemp ve arkadaşları [10], Cuckoo adında bir çerçeve yazılımı ile hesaplama yoğunluklu işleri Android servisleri şeklinde tasarlayarak, bu işleri yerel makine dışındaki makinede yapmıştır. Ön derleme aşamasında uzaktaki bir makineye taşınacak servisler oluşturulup geliştiricinin hesap yoğunluklu işleri bu oluşturulan servisler içinde yazması beklenmektedir. Geliştiricinin yazdığı bu yerel servislerin uzak bilgisayarda çalışacak şekilleri ön-derleme aşamasında otomatik üretilerek çalışma zamanında sunuculara taşınmaktadır. Sunucuya taşınan servislerin metotları aynı da olabilir ama eğer farklı algoritmalar çalıştırılacaksa bu servislerin metotları farklı da olabilir.

Bu yöntem Android Arayüz Tanımlama Dilini kullandığı için ilk aşamada işletim sistemi ile uyumlu olarak aktivitelerdeki metotları geri çağırma özelliğine sahip olsa da bu metotların tekrar sunucuya dönmesi çeşitli zorluklar içermektedir. Bunun yanında tüm hesaplama yoğunluklu servislerin AIDL kullanılarak yazılması gerektiğinden dolayı servis yerelde çalışacak olsa bile yine de Android IPC mekanizması ile servislere bağlanılması gerektiğinden fazladan iletişim ve veri aktarma maliyetleri söz konusu olmaktadır. Diğer taraftan uygulama geliştiriciler için ise hesaplama yoğunluklu işleri belirleyip bunları Android Servisleri şeklinde yazmak gibi ek bir iş yükü getirmek bu yöntemin dezavantajlarıdır.

3.3 R-OSGi Servis Yöntemi

OSGi merkezileştirilmiş Servis Tabanlı Mimariyi (SOA) temel almaktadır. Bu yöntem java sınıflarının bir servis olarak yayınlanmasını sağlayıp diğer paketlerin (bundles) bu sınıfları kullanmasına olanak sağlamaktadır. Bir servis, servislerin içinde yayınlandığı bir uygulama arayüzü ve servis özelliklerini içermektedir. OSGi yapısı sistemde yayınlanan tüm servislerin kayıtlarını tutmaktadır [16]. OSGi terminolojisinde yazılım modülleri paketler (bundles) olarak adlandırılmaktadır ve bu paketler servisler vasıtasıyla birbiri ile iletişim sağlamaktadır. OSGi spesifikasyonları yerel sanal makine referanslarını temel aldığından, bunlar dağıtık uygulama modülleri için tanımlanmamıştır. Rellermeyer ve arkadaşları [16] R-OSGi (Remote -OSGi) olarak adlandırdığı dağıtık OSGi'i uzak servislerin bulunması ve uzak servislerin kayıt edilmesi yöntemini kullanarak yerleştirmiştir, böylece kullanıcıların bu servisleri bulup kullanması mümkün olmuştur. R-OSGi farklı makinalardaki paketlerin iletişimini yönetmesi için yerleştirilmiştir. Yerel bir paket böyle bir uzak servisi çağırmak istediğinde, R-OSGi bir yerel vekil paket üreterek bunun arayüzünü yerel olarak kullanıma sunar ve bu vekil paket çağrıldığında, R-OSGi uzak metot çağırma mekanizması ile uzaktaki paketin servislerinin metotlarını çağırmayı sağlar. Rellermeyer ve arkadaşları [17], AlfredO adında bir yapı geliştirerek etrafımızdaki tüm elektronik aletlerin yazılımlarını önceden kurulmuş sürücüler yerine servis olarak vermesini mümkün kılmıştır. Her bir elektronik alet yeteneklerini, dinamik olarak akıllı telefonların erişebileceği modüler servis çeşitleri şeklinde yayınlamıştır.

4 Vekil Tabanlı Kod Taşıma Yöntemi

Akıllı telefonlarda hesaplama yoğunluklu metotlara sahip nesneleri uzaktaki bir sunucuya götürüp işlemleri orda yapıp sonucu istemciye dönmek bu çalışmanın temel hedefidir. Fakat RMI mekanizmasını akıllı telefonlarda uygulama geliştiricilere ek yük getirmeden ve uzaktaki nesnenin akıllı telefondaki bir yerel nesneye ihtiyaç duyması gibi durumlarda kullanmak çok güç olmaktadır. Uzak Metot Çağırma mekanizmasını, akıllı telefonlar için uygulama geliştiricilere ek bir yük getirmeden ve saydam bir şekilde gerçekleştirmek için yukardaki prensipleri temel alan daha basit bir yöntem geliştirdik. Bizim geliştirdiğimiz yöntemde geliştiriciler nesne yaratma görevini "new" anahtar sözcüğünü kullanma yerine bir fabrika sınıfına vermektedirler yani yaratmak istedikleri tüm nesneleri bu fabrika sınıfından talep etmektedirler.

Bu fabrika sınıfı yaratılacak nesneler için dinamik vekil (proxy) yaratarak geliştiricilere bunu vermektedir ve nesneleri takip edebilmektedir. Böylece yerel nesneler hakkında gerekli geçmiş bilgiler toplanarak uzaktaki bir bilgisayara gönderilecek hesaplama yoğunluklu nesneler belirlenir ve uzak bir makinaya taşınır. Çalışma zamanında akıllı telefondaki bir nesnenin uzaktaki nesnelere ihtiyaç duyması durumunda fabrika sınıfı uzaktaki nesnelerin dinamik vekil sınıflarını vererek uzaktaki bu nesnenin metotlarına erişimi sağlamaktadır. Ulaşım katmanı olarak TCP/IP tabanlı bağlantı prensibi kullanıldı ve iletişim tamamıyla fabrika sınıfı vasıtasıyla geliştiriciden bağımsız bir şekilde yapılmaktadır. Geliştirdiğimiz yöntem ile ilgili detaylı bilgiyi Kaya ve Koçyiğit [18] bildirisinde bulabilirsiniz.

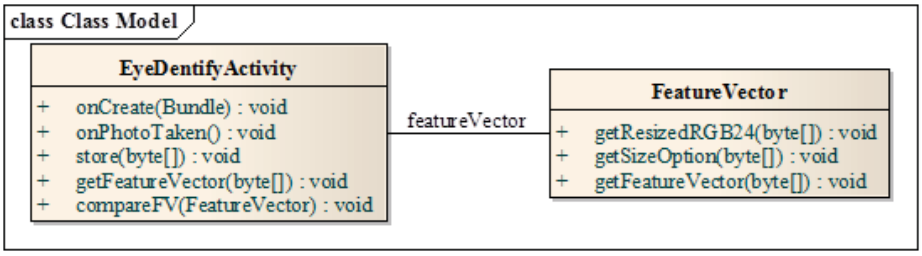
5 Örnek Uygulama ve Performans Değerlendirme

Bu çalışmada, kod taşıma kararı metotların işlem süreleri ve enerji tüketimleri metrikleri temel alınarak gerçekleştirilmiştir. Vekil Tabanlı Kod Taşıma yöntemi performansı, nesne tanıma uygulaması yerleştirilerek değerlendirilmiştir.

Nesne Tanıma Uygulaması: Bu uygulamada, nesne öğrenme ve tanıma olmak üzere iki mod bulunmaktadır. Öğrenme modunda telefon kamerası ile çekilen fotoğrafların özellik vektörleri hesaplanarak nesne ve kullanıcı adı ile kayıt edilir. Tanıma modunda ise çekilen fotoğrafların özellik vektörleri hesaplanarak bu özellik vektörü kayıtlı vektörlerle karşılaştırılır. Bu karşılaştırmada kullanıcının belirlemiş olduğu bir eşik değer temel alınarak en benzer fotoğraf bulunup nesne ve kullanıcı adı ile telefonda gösterilmektedir.

Nesne tanıma uygulamasının kısmi sınıf diyagramı aşağıda şekil 2'de verilmiştir. EyeDentifyActivity nesnesi grafik arayüz sınıfı olup telefonda yerleştirilmiştir ve FeatureVector nesnesi ise hesaplama yoğunluklu bir nesne olup buluta taşıma için işaretlenmiştir. Çalışma zamanında FeatureVector nesnesi yaratıldığı zaman bu nesnenin dinamik vekili yaratılıp verilmektedir. Böylece bu vekil nesne metotları çağrıldığı zaman ağ bağlantısı ile buluta gidilip bu nesnenin metotları çağrılıp metot sonucu dönülmektedir. Aşağıdaki kod satırında, dinamik vekil nesne yaratılması işi Fabrika sınıfına verilişi gösterilmiştir (1).

```
featureVector = (FeatureVector) OffloadingFactory(FeatureVector.class, Param)    (1)
```

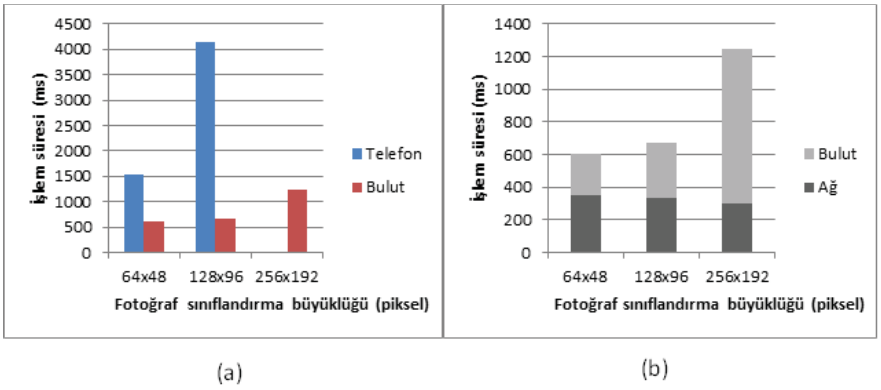


Şekil. 2. Nesne tanıma uygulaması kısmi sınıf diyagramı

5.1 İşlem Süreleri Sonuçları

Bu uygulamayı çeşitli fotoğraf büyüklüklerine ve farklı donanım özelliklerine sahip akıllı telefon markalarında her fotoğraf büyüklüğü için 10 defa çalıştırdık. Kullandığımız akıllı telefonlar ve özellikleri şöyledir: Samsung Galaxy S2 (1 GHz dual-core CPU, Android 4.1.2-Jelly Bean işletim sistemi) ve HTC Evo 3D (1.2 GHz dual-core CPU, Android 4.0.3 ICS işletim sistemi). Bulut altyapısı için kullandığımız bilgisayar özellikleri: 2.0 GHz Intel Core i7 263QM CPU ve 8 GB RAM. 64-bit Windows 7 işletim sistemi. Deneysel çalışmada bulut bilgisayarı, yerel alan ağı ve geniş alan ağı ile erişilmiştir. Akıllı telefonlar yerel alan ağına 54 Mbps kapasitesine sahip ulaşım noktaları ile internete bağlanmışlardır. Bulut bilgisayarı ise 100BaseTX Ethernet ve 100 Mbps bağlantısı ile internete bağlanmıştır.

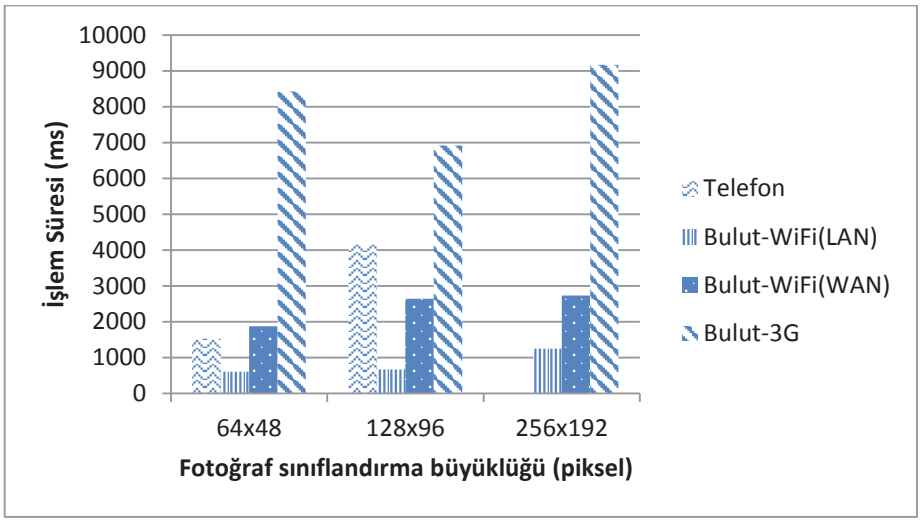
Nesne tanıma uygulamasını 480x800 ve 1232x2048 piksel büyüklüğünde fotoğraflar için ayrı ayrı çalıştırdık. Bu uygulama, özellik vektörünü hesaplamak için verilen fotoğraf büyüklüklerini 64x48, 128x96 ve 256x192 piksel alt fotoğraf parçalarına ayırmaktadır. Yüksek alt fotoğraf büyüklüğü daha iyi sonuçlar vermesine rağmen 256x192 piksel alt fotoğraf büyüklüklerinde akıllı telefonlar, özellik vektörlerini hesaplarken hafıza yetersizliği hatası vermektedir. Şekil 3(a)'da nesne tanıma algoritmasının kullandığı her bir alt fotoğraf büyüklüğü için uygulamayı 10 defa çalıştırıp ortalama işlem sürelerini karşılaştırdık. İlk aşamada aynı yerel ağ üzerinden buluta bağlanarak 480x600 ve 1232x2048 piksellik fotoğraflar gönderdik. Hesaplama yoğunluklu işleri buluta yaptırarak %60 ile %83 oranında işlem süresi tasarrufu sağladık. Burada önemli bir nokta, 256x192 piksellik alt fotoğraf büyüklüklerinde telefonda hafıza yetersizliği problemi ile karşılaştık. Şekil 3(b)'de buluta yaptırılan işlerin ağda ve bulutta geçen işlem sürelerini karşılaştırdık.



(a) (b)

Şekil. 3. İşlem süreleri karşılaştırma (a) Telefon ve bulut işlem sürelerinin karşılaştırılması (b) Ağ ve bulut işlem süreleri detayı

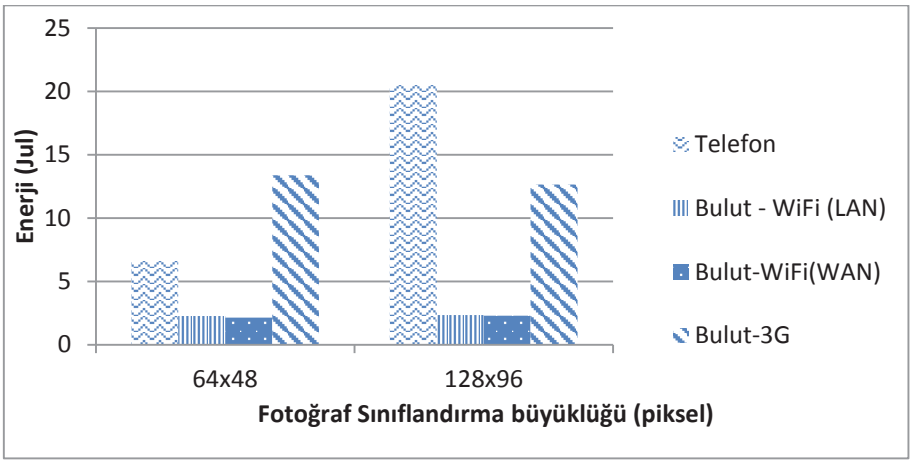
Şekil 4'de görüldüğü gibi yerel ağdaki bir buluta bağlanarak hesap yoğunluklu işlerin yapılması işlem tasarrufu sağlarken uzaktaki bir bilgisayara işlerin yaptırılması ağdaki gecikme yüzünden daha az avantajlı duruma gelmektedir. Bunun yanında uzaktaki bir bilgisayara 3G ile bağlanarak işlerin yaptırılması telefonda yaptırılmaya göre daha dezavantajlı bir sonuç vermektedir.



Şekil. 4. Yakındaki ve uzaktaki bir buluta hesaplama yaptırılması sonucu işlem sürelerinin karşılaştırılması

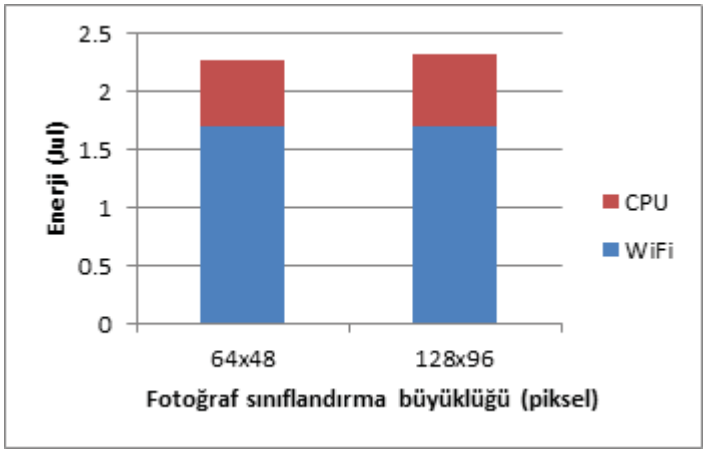
5.2 Enerji Tüketimi Sonuçları

PowerTutor [19] enerji analiz aracını kullanarak akıllı telefonlarda nesne tanıma uygulamasını çalıştırdığımızda hem telefondaki işlem için harcanan enerjiyi hem de internet veri aktarımı için harcanan enerjiyi hesapladık. Bu yazılım aracı HTC Evo akıllı telefonlar için geliştirildiği için enerji tüketim sonuçlarını bu telefon üzerinde yaptık. Her bir fotoğraf için nesne tanıma uygulamasını 50 defa çalıştırarak ortalama enerji tüketim değerlerini bulduk. Nesne tanıma için kullandığımız fotoğraf büyüklükleri 0.4 MP (480x600) ve 2.5 MP (1232x2048)'dir. Şekil 5'de görüldüğü gibi hesaplama yoğunluklu işleri telefonda yaptırmak yerel ağ üzerindeki bir bilgisayarda yaptırmaya göre daha fazla enerji tüketmektedir. Hesaplama yoğunluklu işleri buluta yaptırarak %65 ile %88 oranında enerji tasarrufu sağladık. 3G ile uzaktaki bir bilgisayara işleri yaptırmak yerel ağdaki bir bilgisayarda yaptırmaya göre daha fazla enerji tüketmektedir. Uzaktaki bir buluta işleri yaptırırken tutarlı sonuçlar elde etmek için Traceroute [19] yazılımı ile durak(hop) sayısında değişiklik olan ölçümleri elimine ettik.



Şekil. 5. Enerji tüketimlerinin karşılaştırılması

Şekil 6'da yerel ağdaki bir buluta hesaplama yoğunluklu işlerin yaptırılması durumunda telefonun işlemci ve ağ için enerji tüketimleri verilmektedir. Şekil 6'dan da görüleceği gibi enerji tüketiminin büyük bir kısmı ağ üzerinden veri aktarılması esnasında oluşmuştur



Şekil. 6. Yerel ağdaki bir buluta işlerin yaptırılması aşamasında işlemci ve ağ enerji tüketimlerinin karşılaştırılması

6 Sonuç

Akıllı telefonlar günlük hayatımızın önemli bir parçası olmakla birlikte gelişen donanım teknolojisine paralel olarak yazılım alanında da akıllı telefonlardan beklentiler yükselmektedir. Ama akıllı telefonlar masaüstü ve sunucu bilgisayarlara

nazaran sınırlı işlemci gücü, hafıza ve depolama kapasitesine sahiptirler. Özellikle enerji tüketimi hesaplama yoğunluklu işler olan yazı tanıma, nesne tanıma ve artırılmış gerçeklik gibi yazılımlarda çok yüksek olmaktadır. Bu sorun kullanıcılarda çok büyük memnuniyetsizlik yaratmaktadır. Etrafımızdaki veya uzaktaki hesaplama merkezlerindeki (yerel veya uzak bulut) donanımlarla akıllı telefonları destekleyip hesaplama yoğunluklu işleri saydam ve kesintisiz bir şekilde bulutlara yaptırmak sorunların büyük bir kısmını çözecektir.

Bu çalışmada literatürdeki kod taşıma yöntemi ile hesaplama yoğunluklu işlerin buluta yaptırılması yöntemlerini inceledik. Çalışmaların yoğunlaştığı Arayüz Tanımlama Dili kullanılarak servislerin oluşturulması ile OSGi servisleri şeklinde hesaplama yoğunluklu işlerin yazılması yöntemlerini inceledik. Bu iki yöntem hesaplama yoğunluklu işlerin servisler şeklinde yazılmasını temel almaktadır. Uygulama geliştiricilere getirdikleri ek yüklerin yanında bulutta çalışan bir kod parçacığı telefondaki kaynaklara ihtiyaç duyduğu zaman problemlerle karşılaşmaktadırlar. Bu problemleri Vekil Tabanlı Kod Taşıma yöntemi ile ortadan kaldırıp, işleri buluta saydam ve kesintisiz bir şekilde yaptırarak nesne tanıma uygulaması için %60 ile %83 oranında işlem süresi ve %65 ile %88 oranında enerji tasarrufu sağladık.

7 Kaynaklar

1. Java Uzak Metot Çağırma, http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/, son ziyaret: 05.05.2014.
2. Android Arayüz Tanımlama Dili, http://developer.android.com/guide/components/aidl.html, son ziyaret: 05.05.2014.
3. OSGi mimarisi, http://www.osgi.org/Technology/WhatIsOSGi, son ziyaret: 05.05.2014.
4. ITU. (2013). Measuring the Information Society 2013 (5th Edition ed., Vol. 2013). Geneva, Switzerland: International Telecommunication Union (ITU).
5. Pathak, A., Hu, Y. C., Zhang, M., Bahl, P., and Wang, Y. M., "Enabling automatic offloading of resource-intensive smartphone applications," Purdue University, Tech. Rep. ECE-TR-11-3, 2011.
6. Chun, B. G., Ihm, S., Maniatis, P., Naik, M., and Patti, A., "Clonecloud: elastic execution between mobile device and cloud," In Proceedings of the sixth conference on Computer systems ACM , pp. 301-314, April 2011.
7. Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N., "The case for vm-based cloudlets in mobile computing," Pervasive Computing, IEEE, vol. 8(4), pp. 14-23, 2009.
8. Verbelen, T., Hens, R., Stevens, T., De Turck, F., and Dhoedt, B., "Adaptive online deployment for resource constrained mobile smart clients," InMobile Wireless Middleware, Operating Systems, and Applications, Springer Berlin Heidelberg, pp. 115-128, 2010.
9. Verbelen, T., Simoens, P., De Turck, F., and Dhoedt, B., "AIOLOS: Middleware for improving mobile application performance through cyber foraging," Journal of Systems and Software, 2012.
10. Kemp, R., Palmer, N., Kielmann, T., and Bal, H., "Cuckoo: a computation offloading framework for smartphones," In Mobile Computing, Applications, and Services, Springer Berlin Heidelberg, pp. 59-79 , 2012.

11. Kovachev, D., and Klamma, R., “Framework for Computation Offloading in Mobile Cloud Computing,” International Journal of Interactive Multimedia & Artificial Intelligence, vol. 1(7), 2012.
12. Chen, E., Ogata, S., and Horikawa, K., “Offloading Android applications to the cloud without customizing Android,” In Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on, pp. 788-793, March 2012.
13. Zhang, Y., Liu, H., Jiao, L., and Fu, X., “To offload or not to offload: An efficient code partition algorithm for mobile cloud computing,” InCloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on, pp. 80-86, November 2012.
14. Cuervo, E., Balasubramanian, A., Cho, D. K., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P., “MAUI: making smartphones last longer with code offload,” In Proceedings of the 8th international conference on Mobile systems, applications, and services ACM, pp. 49-62, June 2010.
15. Kristensen, M. D., and Bouvin, N. O., “Scheduling and development support in the scavenger cyber foraging system,” Pervasive and Mobile Computing, vol. 6(6), pp. 677-692, 2010.
16. Rellermeyer, J. S., Alonso, G., & Roscoe, T. (2007, November). R-OSGi: distributed applications through software modularization. In Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware (pp. 1-20). Springer-Verlag New York, Inc.
17. Rellermeyer, J. S., Riva, O., & Alonso, G. (2008, December). AlfredO: an architecture for flexible interaction with electronic devices. In Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (pp. 22-41). Springer-Verlag New York, Inc.
18. Kaya, M., Koçyiğit, A., & Eren, P. E., (2014). A Mobile Computing Framework Based on Adaptive Mobile Code Offloading. In Software Engineering and Advanced Applications (SEAA), 2014, 40th EUROMICRO Conference. IEEE.(Bu bildiri Ağustos 2014’de sunulacaktır)
19. Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., and Yang, L., “Accurate online power estimation and automatic battery behavior based power model generation for smartphones,” In Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis ACM, pp. 105-114, October 2010.
20. Traceroute: Ağ rota bilgisi yazılımı, http://docs.oracle.com/cd/E23824_01/html/821-1453/ipv6-admintasks-72.html, son ziyaret: 05.05.2014.