Durumsal Servislerin Sanallaştırılması

Hasan Ferit Enişer ve Alper Şen

Boğaziçi Üniversitesi, Bilgisayar Mühendisliği Bölümü
{hasan.eniser,alper.sen}@boun.edu.tr

Özet. Günümüzde, kurumsal yazılım sistemleri geçmişe göre çok daha karmaşık hale gelmiştir. Birbirine bağımlı uygulamaların sayısının artması, farklı teknolojilerin aynı anda kullanılması ve bir çok servisin birbiriyle haberleştiği Servis Temelli Mimarilerin yaygın kullanımı bu tür sistemlerin test edilmesini zorlaştırmaktadır. Bu yazılım sistemlerinin test edilmesi konusunda servis sanallaştırma popülerlik kazanmaktadır. Servis sanallaştırma, gerçek bir servisin davranışlarını taklit etmeye yarayan bir tekniktir. Servisler, durumsal ve durumsal olmayan olmak üzere iki sınıfa ayrılabilir. Gerçek hayattaki servislerin bir çoğu durumsaldır. Biz bu çalışmada, durumsal servislerin sanallaştırılması için, sınıflandırma tabanlı ve diziden dizi tahmin eden modeller tabanlı iki farklı çözüm ürettik ve çözümlerimizi gerçek servislerden toplanmış veriler üzerinde test ettik.

Anahtar Kelimeler: Servis Sanallaştırma · Yazılım Mühendisliği · Yazılım Testi.

Stateful Service Virtualization

Hasan Ferit Enişer and Alper Şen

Bogazici University, Department of Computer Engineering
{hasan.eniser,alper.sen}@boun.edu.tr

Abstract. Software systems are becoming more complicated with increasing number of dependent applications, heterogeneous technologies and common usage of Service Oriented Architectures (SOA). This trend makes testing of such systems challenging. For testing these software systems, the concept of service virtualization is an attractive solution. Service virtualization is an automated technique to mimic the behavior of a given real service. Services are in two types namely, stateless or stateful services. Many services are stateful in nature. In this work, we bring a solution to automated stateful service virtualization. We employ classification based and sequence-to-sequence based machine learning algorithms in our solutions. We demonstrate the validity of our approach on two data sets collected from real life services and obtain successful results.

Keywords: Service Virtualization · Software Engineering · Software Testing.

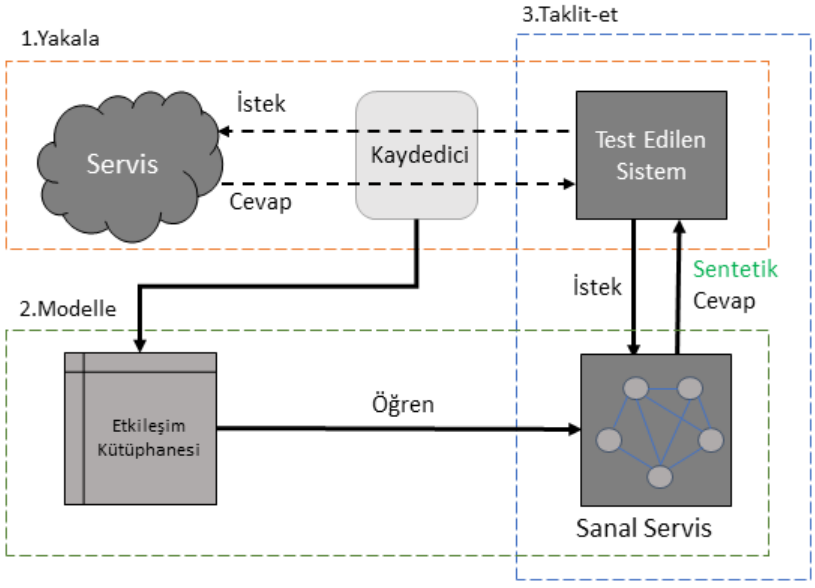
1 Giriş

Günümüz yazılım sistemlerinde, Servis Temelli Mimariler (STM) şirketlere ihtiyaç duydukları esnekliği ve ölçeklenebilirliği sağlamaktadır. Bu tür mimariler birbirine bağlı servislerin ve takımların sayısında artışa sebep olmaktadır. Bunun yanında STM'de birçok farklı teknoloji aynı yazılım sisteminde kullanılmaktadır. Yazılım testçileri ve geliştiricileri aşağıdaki sebeplerden dolayı bağlı oldukları servislere ulaşmakta sıkıntı çekebilmektedirler:

- Henüz geliştirilmesi tamamlanmamış servisler.
- Sınırlı kapasiteli servisler.
- Üçüncü-partiler tarafından sağlanan servislerin maliyeti.
- Aynı anda birden çok geliştirici veya testçi tarafından ihtiyaç duyulan servisler.

Bu sebeplerden dolayı yazılım geliştirmede *test ikizleri*ne ihtiyaç duyulmaktadır. Servis sanallaştırma ile ihtiyaç duyulan servisin sanal bir kopyası yaratılarak, Test Edilen Sistemi (TES) bağımlılıklarından kurtarmak mümkündür. Sanal servisler, gerçek servislerin yerine kullanılarak, özellikle büyük yazılım sistemlerinin testini kolaylaştırmaktadır. Buna ek olarak sanal servisler, gerçek servislerin hem performans hem de veri özelliklerini taklit edebilmektedir.

Servis sanallaştırma Şekil 1'de görüldüğü gibi yakala, modelle, ve taklit-et şekline isimlendirebileceğimiz üç ana adımdan oluşmaktadır. Yakala adımında bir servisi sanallaştırmak için gerekli olan istek-cevap ikilileri kaydedilmektedir. Daha sonra kayıtlı veriden bir model çıkarılır. Son olarak da, sanal servis gerçek



Şekil 1. Servis sanallaştırmaya genel bakış.

servisin yerine görevlendirilir ve gelen isteklere gerçeğe benzer yada gerçekle aynı sentetik cevaplar üretir.

Servisler durumsal ve durumsal olmayan olmak üzere iki sınıfa ayrılabilir. Durumsal servislerde bir istek servisin durumunu değiştirebilir. Durumsal servislerde-cevaplar o andaki duruma bağlı bir fonksiyondan üretilir. Bu tür servislere örnek olarak alışveriş sepeti servisi örneği verilebilir. Kullanıcı sepetin içeriğini görmek ya da ödemeye gitmek istediğinde, servis doğru cevapları üretebilmek için daha önce eklenmiş ya da çıkarılmış maddeleri hesaba katmak zorundadır. Durumsal olmayan servislerde ise cevap üretmek için gerekli olan bütün bilgiler istekte gönderilmektedir, başka bir söyleyişle servisin durumu önemli değildir. Bu tür bir servise örnek olarak bir şehirdeki otellerin isimlerini listeleyen bir servis verilebilir.

Bu bildirideki katkılarımız şöyle sıralanabilir:

- Durumsal servislerin sanallaştırılmasına iki farklı çözüm getirdik.
- Sanal servis yaratmak için makine öğrenmesi metodlarından faydalandık.
- Bu bildiride önerdiğimiz teknikleri bir araç haline getirdik ve gerçek servisler üzerinde doğrulamasını yaptık.
- Bu bildiride önerdiğimiz teknikleri literatürdeki bir durumsal model çıkarma aracıyla karşılaştırdık.

Bu bildirinin ikinci kısmında ilgili çalışmalar, üçüncü kısmında ön bilgiler, dördüncü kısmında önerilen tekniklerin detayları ve beşinci kısımda değerlendirme detayları anlatılmaktadır. Son olarak altıncı kısımda sonuçlar ve gelecek çalışmalar tartışılmıştır.

2 İlgili Çalışmalar

Michelsen ve ark. [17] çalışmasında servis sanallaştırmanın temellerini, detaylarını ve faydalarını anlatmıştır. Servis sanallaştırmanın kullanımına dair bir vaka çalışması [20]'de gösterilmiştir. Servis sanallaştırmaya çözüm getiren son çalışmalar [6, 23, 7, 25, 26] durumsal olmayan servislerin sanallaştırılması için bioinformatik algoritmalarından faydalanmıştır. Bu çalışmalardan sonuncusu [26], eski çalışmalarda elde edilen sonuçları geliştirerek durumsal olmayan servislerin sanallaştırılması için Opaque Service Virtualization (OSV) adıyla bir araç sunmuştur. Enişer ve ark. [9] OSV'de elde edilen sonuçları daha da iyileştirmişlerdir. Bunun dışında [8]'de yazarlar durumsal servislerin sanallaştırılması için literatürde bilinen ilk yöntemleri sunmuştur.

Akademideki çalışmaların yanı sıra, IBM, HP, CA, SmartBear ve Parasoft gibi firmalar kendi servis sanallaştırma araçlarını geliştirmişlerdir. Bu araçların bir karşılaştırması [10, 19]'de bulunabilir. Gerçek kullanıcılar üzerinde, bahsi geçen servis sanallaştırma araçlarıyla alakalı yapılmış bir anket bulunmaktadır [13].

Bunun dışında kara-kutu durum modeli çıkarma teknikleri [2, 3, 22, 21, 15, 16, 27] sanallaştırılmak istenen servisin modelini çıkarmak için uygun bir yöntem

gibi görünmektedir. Fakat daha önceki çalışmaların bu problem özelinde uygulanması için problemin başka bir şekilde formülize edilmesi gerekmektedir. Fakat deneylerde gösterdiğimiz gibi bu işlem yüksek bir maliyet getirmektedir ve sonuçlar memnun edici olmamaktadır.

3 Yöntem

Bu kısımda, örnek bir durumsal servis anlattıktan sonra durumsal servislerin sanallaştırılması için iki farklı yöntem sunacağız. Bu yöntemlerde bir servise gelen herhangi bir istek için doğru cevabı tahmin etmeye çalışacağız. Şekil 1'de gösterildiği gibi daha önce kaydedilmiş istek-cevap dizilerinden öğrenme yoluyla bir model oluşturacağız ve bu modeli gelen isteklere cevap sentezlemek için kullanacağız. İlk olarak bahsi geçen modeli sınıflandırma yöntemiyle öğrenmeyi daha sonra ise derin öğrenme tekniklerini kullanarak öğrenmeyi göstereceğiz. Sınıflandırma tabanlı yöntemde sentezlenmeye çalışılan cevabın her bir içeriği bir sınıf olarak düşünülmüş ve bu içerikler tahmin edilmeye çalışılmıştır. Derin öğrenme tabanlı yöntemde ise modeli eğitirken direk olarak istekleri ve onlara karşılık gelen cevapları verdik ve böylece modelin istekten cevaba çeviri yapmayı öğrenmesini sağladık.

3.1 Örnek Bir Durumsal Servis

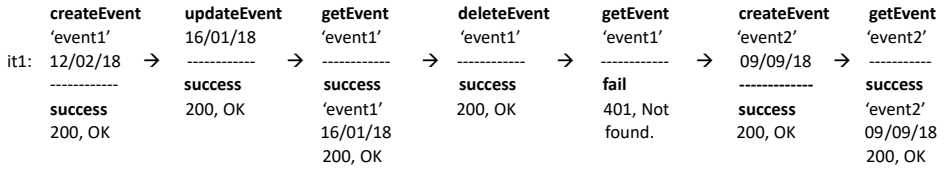
Bu kısımda durumsal servis örneği olarak takvim servisini anlatacağız. Bir kullanıcı takvim servisini kullanarak, etkinlik yaratabilir, etkinlik silebilir, bir etkinliğin detaylarını görebilir ya da etkinliğin bilgilerini güncelleyebilir. Böyle bir servise ait istek-cevap dizileri Şekil 2'de görülebilmektedir. Bu dizide istekler kesikli çizgilerin üzerinde, cevaplarsa altında görünmektedir. İstek tipleri (createEvent, updateEvent vs.) ve cevap tipleri (success, fail) kalın olarak yazılmıştır. Geriye kalan kısımlar ise istek ve cevapların içeriklerini oluşturmaktadır.

Bu diziye göre ilk önce bir etkinlik yaratılmış (createEvent), sonra etkinlik tarihi güncellenmiş (updateEvent) daha sonra da etkinlik detayları istenmiştir (getEvent). Sonrasında etkinlik silinmiş (deleteEvent) ve tekrar detayları isteniştir (getEvent). Fakat bu kez etkinlik silindiği için cevap değişmiş ve hata alınmıştır. Devamında da başka bir etkinlik yaratılmış (createEvent) ve onun detayları istenmiştir (getEvent).

Bu dizi durumsal bir serviste doğru cevabın sentezlenebilmesi için, sadece gelen isteğin değil gelen istekle beraber geçmişin de hesaba katılması gerektiğini göstermektedir. Çünkü örnekte görüldüğü gibi geçmişteki istek-cevap ikilileri yeni gelen bir isteğin cevabını değiştirebilmektedir.

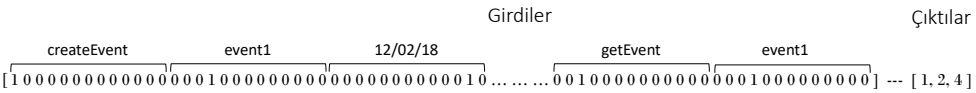
3.2 Sınıflandırma Tabanlı Sanallaştırma (STS)

Sınıflandırma, girdiler ve çıktılar arasındaki ilişkiyi öğrenmek için kullandığımız bir makine öğrenmesi yöntemidir. Sınıflandırmada verilen bir girdi mümkün olan sınıflardan birisine atanır.



Şekil 2. Takvim servisine ait örnek bir istek-cevap dizisi.

Durumsal bir serviste bir isteğin cevabı geçmişteki istek-cevap ikililerinden etkilenebilmektedir. Dolayısıyla durumsal servis sanallaştırma için bir model eğitirken verilen bir isteğin kendisi ve geçmiş istek-cevap ikilileri ile bu isteğin cevabı arasındaki ilişikiyi öğrenmek isteriz. Gerçek hayattaki servislerin istek tipleri, içerikleri ve cevapları çoğunlukla kategorik veri özelliği gösterir. Kategorik verilerde iki veri noktası arasındaki mesafe bütün veri ikilileri için eşittir (Örn. İstanbul, Ankara, İzmir). Bu tür bir veriyi bir makine öğrenmesi algoritmasına girdi olarak verirken *one-hot encoding* yöntemiyle yeniden formatlamak gerekir. Şekil 3'de örnek bir istek-cevap geçmişi ve ona karşılık gelen sınıf atamaları görülmektedir.



Şekil 3. Takvim servisine ait bir veri noktasında örnek bir *one-hot encoding* uygulaması.

STS yöntemi ham isteklerin ve cevapların istek tipi, istek içeriği, cevap tipi ve cevap içeriği şeklinde çözümlenmesini gerektirmektedir. Girdi kısmında, çözümlenen her bir alana *one-hot encoding* uygulanır, çıktı kısmındaysa tahmin edilmesi beklenen cevap içerikleri numaralandırılır. Sonuç olarak bu şekilde bir veri noktası oluşturulmuş olur.

Örnekte görüldüğü gibi eğer birden fazla sınıf tahmin edilecekse ve her bir tahmin için ikiden fazla sınıf ihtimali varsa bu tür bir sınıflandırma problemine çoklu çıktılı çoklu sınıflandırma problemi ismi verilir.

Biz bu çalışmada, bahsedilen sınıflandırma problemi için kural tabanlı bir sınıflandırma yöntemi olan Repeated Incremental Pruning to Produce Error Reduction (RIPPER) algoritmasını [5] kullandık. RIPPER sınıf ayrıştırması için *if-then* kuralları öğrenir ve sınıf atamalarını bu kurallara göre yapar. Farkedileceği gibi doğru *if-then* kurallarını öğrenmek servis sanallaştırma probleminde doğru cevapları üretmek için uygun bir yöntemdir. Dolayısıyla RIPPER burada en uygun sınıflandırma çözümü olarak durmaktadır. Bunun dışında RIPPER'ın eğitme süresi de diğer sınıflandırma algoritmalarına göre (örn: yapay sinir ağları) oldukça iyidir.

Model, eğitilmesi tamamlandıktan sonra gelen bir isteğe cevap üretmek için kullanılmaya hazırdır. Modele yeni bir istek ulaştığında, bu istek, geçmiş istek-cevap ikilileriyle birlikte *one-hot encoding* kullanılarak modele uygun hale getirilir. Eğer bir isteğin geçmişinde veya kendisinde daha önce kaydedilmemiş bir içerik varsa bu içerik bütün diğer istek tipi, içerik vs.den farklı bir şekilde *one-hot encoding* yapısına çevirilir. Bir istek geçmişiyle birlikte bu şekilde modele girdi olarak verildikten sonra, model ilgili cevabı tahmin edecektir.

Bu yöntemi JSON, XML gibi çok kullanılan mesaj formatlarıyla haberleşen servislerde kullanmak daha uygun olacaktır. Çünkü özel bir mesaj formatıyla haberleşen servislerden kaydedilen mesajların çözümlenmesi oldukça zor olabilir ve her bir özel mesaj formatı için ayrı bir çözümleyici gerekmektedir.

3.3 Derin Öğrenme Tabanlı Sanallaştırma

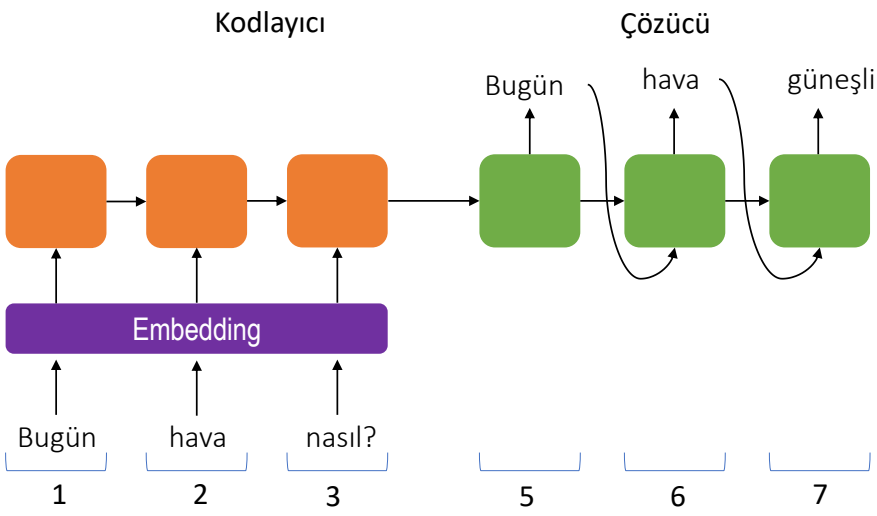
Bu kısımda, durumsal servislerin sanallaştırılması için derin öğrenme tabanlı bir yöntem sunacağız. Derin sinir ağları kavramı özellikle çok katmanlı ve farklı mimarilerdeki yapay sinir ağları için kullanılmaktadır. Derin öğrenme ise derin sinir ağlarının kullanıldığı öğrenme yöntemlerine denir. Biz durumsal servis sanallaştırma için Long Short Term Memory (LSTM) [12] yapay sinir ağı mimarisi ile *diziden-diziye* ismi verilen bir model çeşidi eğittik. Diziden-diziye modelleri girdi olarak bir söz dizisi alır ve çıktı olarak başka bir söz dizisi verir. Bu yöntem özel mesaj formatları için kullanılabilir çünkü dizilerin çözümlenmesi gerekmez.

Diziden-diziye modelleri girdinin bir dizi olarak ele alınması gerektiği çeviri [24] ve otomatik chat-bot yaratma [28, 29] gibi konularda başarılı sonuçlar vermektedir. Diziden-diziye modellerinde devirli sinir ağlarının özel bir formu olan LSTM'ler [12] kullanılmaktadır. LSTM'ler geçmişteki bir bilginin daha sonraki adımlarda kullanılmasına olanak veren bir mimariye sahiptir. Bu özellik durumsal servis sanallaştırma açısından önemlidir çünkü doru cevabı üretmek için geçişteki istek-cevap ikililerini de hesaba katmak gerekir.

Diziden-diziye modellerinin genel yapısı Şekil 4'de görülmektedir. Bu yapı *embedding* adımı ile birlikte bir kodlayıcı ve bir çözücüden oluşur. Kodlayıcı ve çözücü birer LSTM ağıdır.

DTS'ya ilk olarak bir sözlük oluşturarak başladık. Bu sözlüğü Etkileşim Kütüphanesi'ndeki kayıtlı istek ve cevaplardan oluşturduk. Daha önceki çalışmalar [28, 24] sözlüğü kelime bazında oluştururken biz bu sözlüğü elimizde bir çözümleyici olmadığını varsaydığımız için karakterler arasından oluşturduk. Diziden-diziye modellerinde ilk olarak sözlükteki karakterle numaralanır ve girdiler kodlayıcı girmeden önce yeni bir formata çevirilir. Yeni bir formata girdikten sonra girdi kodlayıcıya verilir ve çıktı olarak bu girdiyi temsil eden belli uzunlukta bir yöney elde edilir. Çözücü bu yöneyi alarak tekrardan girdiye karşılık gelen bir diziye çevirir.

Biz durumsal servislerin sanallaştırmasında gelen bir isteğin geçmişini ve bu isteğe karşılık gelen cevabın arasındaki ilişkiyi bulması için bir diziden-diziye modeli eğittik. Bu eğitim yapılırken başarımın artması için belli teknikler uygulamak mümkündür. Örneğin biz bu çalışmada, geçmişin tamamını tek bir seferde



Şekil 4. Diziden-diziye modellerinin genel yapısı kodlayıcı ve çözücü adlı iki kısımdan oluşur. *Bugün hava nasıl?* dizisi *Bugün hava güneşli* dizisine çevirilmiş.

vermek yerine geçmişteki her bir istek için yeni bir veri noktası oluşturup bunları da modelin eğitiminde kullandık. Bu sayede başarımımızın arttığını gördük.

4 Değerlendirme

Bu kısımda, bu bildiride sunduğumuz durumsal servis sanallaştırma yöntemlerinin gerçek servislerin yerine geçmede ne kadar başarılı olduklarını gösterdik ve eğitme sürelerini karşılaştırdık. Kısacası deneyler ile (1) yöntemlerimizin durumsal servis sanallaştırmada ne kadar etkili olduklarını ve (2) yöntemlerimizin hızının ne olduğunu araştırdık. Bunun dışında yöntemlerimizi literatürdeki bir kara-kutu model çıkarma algoritması (EFSM aracı [27]) ile karşılaştırdık.

4.1 Değerlendirmede Kullanılan Servisler

Yöntemlerimizi değerlendirmek için gerçek hayattan iki tane servisi ele aldık. Bu servisler sırasıyla özel bir servis olan Havayolu Biletleme Servisi (HBS) herkesin kullanımına açık olan Google Calendar API'dır (Takvim). Takvim servisi JSON formatıyla haberleşirken, HBS özel bir mesaj formatıyla haberleşmektedir. Her iki servisin de gerçek hayattan alınmış olması ve farklı mesaj formatlarıyla haberleşmeleri bu servisleri seçmemizde etkili olmuştur. HBS 26 farklı istek tipi içerirken, Takvim servisinde 5 farklı istek tipi kullanılmıştır. Değerlendirme aşamasında STS yöntemini HBS servisi üzerinde denemek için özel bir çözümleyici yazdık. HBS'den toplanan istek-cevap ikilileri bir havayolunun biletleme operasyonlarında kullandığı istekler ve cevaplara karşılık gelmektedir. Takvim servisi için bir Python betiği vasıtasıyla rastgele istekler göndererek veri topladık. Toplamda her bir servis için her biri 10 istek-cevap ikilisi içeren 400 istek-cevap dizisi topladık. Deneylerimiz bu sayıların yeterli olduğunu gösterdi.

4.2 Deney Kurgusu

Ölçütler: STS ve DTS yöntemlerini hem doğruluk hem de performans açısından değerlendirdik. Burada performans modellerin eğitim süresine karşılık gelmektedir. STS yönteminin doğruluğunu ölçmek için Kesin Eşleşme Ölçütü (KEÖ) ve Altküme Eşleşme Ölçütü (AEÖ) [18] olmak üzere iki farklı ölçüt hesapladık. DTS yöntemi için ise Kesinlik ölçütünü hesapladık. Yöntemlerimizin doğası gereği ikisi için de aynı ölçütü kullanmak mümkün değildir. Fakat Kesin Eşleşme Ölçütü ve Kesinlik birbiriyle karşılaştırılabilmektedir. İki yöntemin karşılaştırması bu ölçüt üzerinden yapılabilir.

Kesin Eşleşme Ölçütü'nde bir tahminin doğru sayılabilmesi için modelin çıktısındaki sınıf atamalarının tamamının doğru olması gerekir. Görüldüğü gibi bu ölçüt tahminlerdeki kısmen doğru yapılan sınıf atamalarını göz ardı etmektedir. Buna karşın Altküme Eşleşme Ölçütünde bir tahminin doğruluğu model çıktısındaki doğru sınıf atamaları toplam sınıf atamasına bölünmesiyle bulunur. Örneğin doğru çıktı [1, 6, 3] iken eğittiğimiz modelin çıktısı [1, 2, 3] ise Kesin Eşleşme Ölçütünde bu bu çıktının doğruluğu 0 olarak hesaplanırken Altküme Eşleşme Ölçütünde doğruluk $\frac{2}{3}$ olarak hesaplanır.

Bunun dışında deneylerimizde STS yöntemi için mikro- ve makro-ortalamalı F-skorunu da hesapladık. F-skor temelde ikili sınıflandırma için tanımlanmıştır. Diğer bir deyişle basit F-skor atama yapılabilecek sadece iki sınıf varken kullanılır. Fakat bunun skorun çoklu-sınıf sınıflandırma problemleri için başka bir versiyonu mevcuttur. Biz de bu çalışmada çoklu-sınıflar için önerilmiş F-skor versiyonunu kullandık.

Yukarıda bahsettiğimiz gibi DTS'de doğruluğu ölçerken Kesinlik ölçütü kullanılır. DTS'nin çıktısı karakterlerden oluşan bir dizidir. Kesinlik ölçütüne göre eğer bu dizi beklenen çıktı ile tam olarak uyumlu ise doğru sayılır. Yanlızca bir karakterin bile yanlış olması çıktının yanlış olarak değerlendirilmesine sebep olur.

Deneylerin Tasarımı: Deneylerimizde, her bir sanallaştırma yöntemiyle farklı uzunluklarda (k=1, k=5 ve k=10) geçmiş istek-cevap dizilerinin hesaba katıldığı üç farklı sanal servis oluşturduk. k=1 için sadece son istek-cevap ikilisini hesaba katarken k=5 (k=10) için son 5 (10) istek-cevap ikilisini ele aldık.

STS yönteminde faydalandığımız RIPPER yöntemi için Weka makine öğrenmesi kütüphanesini [11] kullandık. Weka içinde bir çok algoritmanın hazır olarak bulunduğu birleşik bir veri işleme aracıdır. Bunun dışında STS'yi kullanırken HBS için bir çözümleyici yazmamız gerekiyordu, fakat, Takvim servisi için Python'un kendisinde bulunan JSON çözümleyicisini kullandık.

DTS yöntemiyle sanallaştırma yaparken Tensorflow [1] kütüphanesinden faydalandık. Fakat Tensorflow'u direk kullanmak yerine Keras'tan [4] faydalandık. Keras, Tensorflow gibi farklı makine öğrenmesi kütüphanelerinin üzerinde çalışan yüksek-düzeyli bir sinir ağı API'sidir. DTS yöntemini kullanarak sanallaştırılacak her bir servis için 1, 5 ve 10 uzunluğundaki geçmiş istek-cevap dizilerini kullanarak 3 farklı sanal servis oluşturduk. Eğittiğimiz diziden-diziye modeli 1 kodlayıcı, 2 çözücü ve 1 tane de ek katman olmak üzere toplam 4 gizli katmandan oluşmaktadır. Kodlayıcı ve çözümleyici katmanları 128'er nörondan

oluşmaktadır. Eğitim sırasında girdileri 128'erlik gruplar halinda verdik ve toplamda 1000 tekrarlama ile eğitimi tamamladık. Modelin eğitimi için ADAM oprimizasyon yöntemini [14] kullandık ve kayıp fonksiyonu olarak kategorik ters entropiden faydalandık.

EFSM aracını internetten indirerek kullandık ve deneylerimizde mümkün olan sınıflandırma parametreleri arasından J48'i seçtik. Bu parametre EFSM aracının en iyi çalıştığı parametre olarak gösterilmiştir [27]. J48 seçeneği bir çeşit karar ağacı çıkarmaktadır. Bu aracı kullanmak için servis sanallaştırma problemini bir model çıkarma problemi olarak formüle ettik.

Deneylerimiz her bir servis için üç ana adımdan oluşmaktadır:

1. Gerçek servisten istek-cevap ikililerini toplama
2. Bu ikilileri kullanarak bir model öğrenme
3. Öğrenilen modelin doğruluğunu ve performansını ölçme

Bunun dışında deneylerde STS yöntemi ve EFSM aracı için 5-katlı çapraz geçerlilik uyguladık ve deneylerimizi 16 GB hafızalı ve Intel Xeon E5-2630L v2 2.40GHz CPU işlemcili bir sunucuda gerçekleştirdik.

4.3 Deney Sonuçları

Bu kısımda yaptığımız deneylerin sonuçlarından bahsedeceğiz. Doğruluk ve performans deneylerimizin sonuçarı Tablo 1 ve Tablo 2'da görülmektedir.

Tablo 1. STS ve DTS yöntemlerinin farklı k değerleri için doğruluk sonuçları. k sayısı hesaba katılan geçmiş istek-cevap dizisinin uzunluğudur.

Servis	k	STS				EFSM Aracı				DTS
		KEÖ(%)	AEÖ(%)	F_{macro}	F_{micro}	KEÖ(%)	AEÖ(%)	F_{macro}	F_{micro}	Kesinlik
HBS	1	76.6	79.6	0.781	0.767	78.1	80.6	0.803	0.783	92.1
Takvim	1	70.3	78.5	0.757	0.741	70.7	76.0	0.721	0.715	93.2
HBS	5	82.7	84.3	0.813	0.798	80.0	83.7	0.806	0.786	96.5
Takvim	5	81.1	84.1	0.843	0.825	71.5	77.1	0.753	0.741	97.3
HBS	10	82.7	84.3	0.813	0.798	80.0	83.7	0.806	0.786	96.5
Takvim	10	82.0	88.5	0.871	0.866	72.1	78.0	0.767	0.751	99.3

Doğruluk Sonuçları: Tablo 1 görüldüğü üzere bütün ölçütlerde en yüksek doğruluk oranları her zaman k=10 için elde edilmiştir. Bu durum her iki servis için de geçerlidir. F-mikro ve F-makro değerleri deneylerde alınan görece yüksek KEÖ, AEÖ ve kesinlik ölçütlerini desteklemektedir.

Bunun dışında, STS kullandığımızda HBS'de Takvim servisine göre daha iyi sonuçlar elde ettiğimizi gördük. Bunun sebebi de HBS'nin sınırlı sayıda içerik veya cevap içeriğinin olmasıdır. Buna karşın Takvim servisinde çok farklı sayıda içerik bulunmaktadır.

Tablo 1'in gösterdiğine göre en iyi sonuçlar DTS ile elde edilmiştir. Bu durum her iki servis için de geçerlidir. DTS tekniği ile elde edilen sonuçlar her zaman

(KEÖ göz önünde bulundurulduğunda) STS ile elde edilenden iyi olmuştur. En yüksek doğruluk oranıysa k=10 için alınmıştır. DTS yöntemi k=10 için hem HBS hem de Takvim servisinde %100'e yakın bir başarı elde etmiştir. Veriye daha detaylı baktığımızda %100'e ulaşmanın imkansız olduğunu farkettik. Bunun sebebi bazı cevapların geçmiş istek-cevap ikililerinde bulunan bilgilerden tamamen farklı içeriklerden oluşuyor olmasıdır. Örneğin geçmiş istek-cevap ikililerinden bağımsız olarak cevapta o günün saat ve tarihi bulunabilmektedir.Bu tür bir cevabı tahmin etmek için bu bildirideki yöntemleri kullanmak gerçekçi değildir. Fakat bu tür bir problem kullanıcı tarafından küçük bir manuel eforla halledilebilir.

EFSM aracının sonuçlarına baktığımızda, değerlerin genellikle STS ile birbirine yakın olmasına rağmen çoğu zaman STS'den düşük olduğunu görmekteyiz.

Özetlemek gerekirse, DTS kullanılarak oluşturulan sanal servisler %90'ın üzerinde bir doğrulukla çalışabilmektedir. STS kullanılarak oluşturulan servislerse daha düşük doğruluk oranlarına sahiptir dolayısıyla daha düşük eşleşme ölçütünün yeterli olduğu durumlarda kullanılabilir.

Tablo 2. STS ve DTS yöntemlerinin farklı k değerleri için performans sonuçları. k sayısı hesaba katılan geçmiş istek-cevap dizisinin uzunluğunu göstermektedir. Tabloada gösterilen eğitim süresi şu formattadır: (hh:mm).

Servis	k	DTS	STS	EFSM Aracı
HBS	1	01:42	00:01	00:06
Takvim	1	02:21	00:01	00:06
HBS	5	08:51	00:03	00:11
Takvim	5	09.54	00:03	00:14
HBS	10	14:42	00:04	00:18
Takvim	10	16:19	00:04	00:19

Performans Sonuçları: Tablo 2 görüldüğü gibi STS yöntemiyle sanal servis oluşturmak DTS yöntemine göre çok daha hızlıdır. RIPPER sınıflandırma algoritmasının kullanıldığı STS yöntemi dakikalar içerisinde eğitilirken, DTS yöntemiyle sanal servis oluşturmak saatler almaktadır. Bu genel olarak eğitmesi zor olan derin sinir ağlarını kullanmanın bir sonucudur. Fakat DTS'nin performans sonuçlarının iyileştirmenin CPU yerine GPU kullanarak mümkün olabileceğini düşünmekteyiz. EFSM aracı ise DTS'den çok daha hızlı çalışıyor olmasına rağmen STS'den daha yavaş çalışmaktadır.

Sonuç olarak eğer eğitme süresi önem arzediyorsa STS yöntemini kullanmak mantıklı olacaktır. Diğer yandan sanal servisin oluşturulmasında doğruluk daha önemli ise DTS yöntemiyle oluşturmak daha iyi sonuçlar verecektir.

5 Sonuç

Yazılımda Servis Temelli Mimariler şirketlere bir çok fayda sağlasa da yazılım geliştirme ve yazılım testinde yüksek karmaşıklıklıktan dolayı sıkıntılara sebep

olabilmektedir. Biz bu çalışmada yazılım bileşenlerinin birbirine bağımlılıklarından dolayı ortaya çıkacak sorunların ortadan kaldırılması için otomatik servis sanallaştırma yöntemleri geliştirdik. Yöntemlerimiz sanallaştırması çok daha zor olan durumsal servisler üzerinde çalışabilmektedir. Yöntemlerimizde makine öğrenmesi algoritmalarından faydalandık. Sunduğumuz yöntemlerin birincisinde (STS) servis sanallaştırma problemini bir sınıflandırma problemi olarak formülize ettik. İkinci yöntemde ise (DTS) diziden-diziye modellerini yani LSTM yapısındaki yapay sinir ağlarını kullanarak sanallaştırma yaptık. Deneylerimizde yöntemlerinizin gelen isteklere doğru cevapları üretmede başarılı olduğunu ve EFSM aracıyla karşılaştırıldığında hem doğruluk hem de performans açısından daha iyi sonuçlar verdiğini gösterdik. STS yönteminin eğitme süresi açısından daha iyi sonuç verdiğini, DTS yönteminin ise doğruluk açısından en başarılı performansı gösterdiğini gördük. Bunun dışında dizi uzunluğunun doğru cevap üretmede oldukça önemli olduğunu gördük. Gelecekte istek-cevap ikililerini kullanarak servislerin sonlu durum makinesini çıkarmayı planlıyoruz. Çıkardığımız sonlu durum makineleri servis sanallaştırma probleminin yeniden formülize edilmesini gerektirmeyecektir ve bu makinelerin doğruluğu yüksek olacaktır. Bunun yanında halihazırdaki yöntemlerimizin doğrulukla beraber performansını da artırmayı düşünüyoruz.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
2. Beschastnikh, I., Brun, Y., Ernst, M.D., Krishnamurthy, A.: Inferring models of concurrent systems from logs of their behavior with csight. In: International Conference on Software Engineering (2014)
3. Beschastnikh, I., Brun, Y., Schneider, S., Sloan, M., Ernst, M.D.: Leveraging existing instrumentation to automatically infer invariant-constrained models. In: ACM SIGSOFT symposium and European conference on Foundations of software engineering (2011)
4. Chollet, F., et al.: Keras. https://github.com/fchollet/keras (2015)
5. Cohen, W.W.: Fast effective rule induction. In: 12th International Conference on Machine Learning (1995)
6. Du, M., Schneider, J.G., Hine, C., Grundy, J., Versteeg, S.: Generating service models by trace subsequence substitution. In: International ACM Sigsoft conference on Quality of software architectures (2013)
7. Du, M., Versteeg, S., Schneider, J.G., Han, J., Grundy, J.: Interaction traces mining for efficient system responses generation. SIGSOFT Software Engineering Notes **40**(1) (2015)
8. Enişer, H.F., Sen, A.: Testing service oriented architectures using stateful service visualization via machine learning. In: Proceedings of the 13th International Workshop on Automation of Software Test. pp. 9–15. ACM (2018)
9. Eniser, H.F., Sen, A., Polat, S.O.: Fancymock: creating virtual services from transactions. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing. pp. 1576–1578. ACM (2018)
10. Giudice, D.L.: Service virtualization and testing solutions. Forrester Wave (2014)

11. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD Explorations **11**(1), 10–18 (2009)
12. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
13. IT-Central-Station: Service virtualization a peek into what real users think. https://goo.gl/oN23qH (2017), accessed at December 2017
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
15. Krka, I., Brun, Y., Medvidovic, N.: Automatic mining of specifications from invocation traces and method invariants. In: ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 178–189. ACM (2014)
16. Lorenzoli, D., Mariani, L., Pezzè, M.: Automatic generation of software behavioral models. In: International conference on Software engineering. pp. 501–510. ACM (2008)
17. Michelsen, J., English, J.: What is service virtualization? In: Service Virtualization, pp. 27–35. Springer (2012)
18. Mohammad, S.S.: A literature survey on algorithms for multi-label learning. Oregon State University, Corvallis **73** (2010)
19. Murphy, T.E., Wilson, N.: Magic quadrant for integrated software quality suites. Gartner Research (2013)
20. Nizamic, F., Groenboom, R., Lazovik, A.: Testing for highly distributed service-oriented systems using virtual environments. Dutch Testing Day (2011)
21. Ohmann, T., Herzberg, M., Fiss, S., Halbert, A., Palyart, M., Beschastnikh, I., Brun, Y.: Behavioral resource-aware model inference. In: International conference on Automated software engineering (2014)
22. Ohmann, T., Thai, K., Beschastnikh, I., Brun, Y.: Mining precise performance-aware behavioral models from existing instrumentation. In: International Conference on Software Engineering. pp. 484–487. ACM (2014)
23. Schneider, J.G., Mandile, P., Versteeg, S.: Generalized suffix tree based multiple sequence alignment for service virtualization. In: Australasian Software Engineering Conference (ASWEC) (2015)
24. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in neural information processing systems (2014)
25. Versteeg, S., Du, M., Bird, J., Schneider, J.G., Grundy, J., Han, J.: Enhanced playback of automated service emulation models using entropy analysis. In: International Workshop on Continuous Software Evolution and Delivery (CSED) (2016)
26. Versteeg, S., Du, M., Schneider, J.G., Grundy, J., Han, J., Goyal, M.: Opaque service virtualisation: a practical tool for emulating endpoint systems. In: International Conference on Software Engineering Companion (2016)
27. Walkinshaw, N., Taylor, R., Derrick, J.: Inferring extended finite state machine models from software executions. Empirical Software Engineering **21**(3), 811–853 (2016)
28. Xu, A., Liu, Z., Guo, Y., Sinha, V., Akkiraju, R.: A new chatbot for customer service on social media. In: Conference on Human Factors in Computing Systems (2017)
29. Zhou, X., Hu, B., Chen, Q., Tang, B., Wang, X.: Answer sequence learning with neural networks for answer selection in community question answering. arXiv preprint arXiv:1506.06490 (2015)