Otomatik Sanal Servis Oluşturma

Hasan Ferit Enişer[1], Alper Şen[1], ve Süleyman Olcay Polat[2]

[1] Boğaziçi Üniversitesi, Bilgisayar Mühendisliği Bölümü, İstanbul, Türkiye
hasan.eniser@boun.edu.tr, alper.sen@boun.edu.tr
[2] Netaş Telekom, İstanbul, Türkiye sopolat@netas.com.tr

Özet. Servis Odaklı Mimariler (SOM), API ve bulut tabanlı uygulamalar, günümüz yazılım dünyasında oldukça yaygın olarak kullanılmaktadır. Bu tür mimarilerin test edilmesi yüksek maliyet, bağımlı servisin uygun veya hazır olmaması ve fazla kapasite kullanımı gibi sebeplerden dolayı zordur. Sanal servisler bu tür durumlarda test ekibi tarafından gerçek servisin davranışını taklit etmek için kullanılır. Sanal servisler, servis arayüzü tanımları (WSDL) veya kaydet-oynat yapısı kullanılarak yaratılabilir. Bunun haricinde sanal servisin davranışı manuel olarak da tanımlanabilir. Otomatik ve akıllı servis sanallaştırma yöntemleri ile ilgili yeterince çalışma bulunmamaktadır. Bu bildiride FancyMock adını verdiğimiz kaydet-oynat yapısıyla çalışan, bioinformatik ve makine öğrenmesi algoritmalarından faydalanan bir servis sanallaştırma aracını geliştirdik. Bu aracın yarattığı sanal servisler daha önce kaydedilmemiş istekler için dahi geçerli ve mantıklı sentetik cevapları pratikte geçerli olabilecek bir süre içerisinde üretebilmektedir. Yaklaşımımızın geçerliliğini gerçek servislerden topladığımız farklı veri setleri üzerinde gösterdik.

Anahtar Kelimeler: Servis Odaklı Mimariler, Yazılım Testi, Servis Sanallaştırma

Automatic Virtual Service Creation

Hasan Ferit Enişer[1], Alper Şen[1], ve Süleyman Olcay Polat[2]

[1] Boğaziçi University, Computer Engineering Department, Istanbul, Turkey
hasan.eniser@boun.edu.tr, alper.sen@boun.edu.tr
[2] Netaş Telecommunications, Istanbul, Turkey sopolat@netas.com.tr

Abstract. Service Oriented Architectures (SOA), API and Cloud based applications are widely used in industry. Testing such complicated systems can be challenging due to multiple reasons including unavailability of components, high cost of using services or high overhead of transactions. Virtual services are employed to simulate the response behaviour of the real service to oversome these challenges. Virtual services are created

by analyzing service specifications (such as WSDL), by recording and replaying transactions, or by determining the behavior manually. There is currently a lack of studies that presents intelligent and automated methods for service virtualization. In this paper, we develop FancyMock tool which is a service virtualization tool that makes use of bioinformatics and machine learning algorithms. Our virtual services can synthesize valid and logical responses in an acceptable amount of time in practice. We show the validity of our approach on three different data sets collected from real services and obtain promising results.

Keywords: Service Oriented Architectures, Software Testing, Service Virtualization

1 Giriş

Günümüzde yazılım sistemleri uygulamaların ve etkileşimlerin artmasıyla gittikçe daha karmaşık bir hal almaktadır. Bunun yanında Continuous Integration (CI) gibi kavramlar de gün geçtikçe popülerleşmektedir. Bütün bu gelişmeler sistemin bütün parçalarıyla test edildiği entegrasyon testini zorunlu hale getirmektedir. Fakat gerçek hayatta entegrasyon testinin önünde bazı zorluklar bulunmaktadır [12]. Bu zorluklar arasında ihtiyaç duyulan servislerin tamamlanmamış olması, servisin sınırlı kapasitede olması, servisin ücretli olması, ve sınırlı izinle kullanılabilmesi gibi sebepler sayılabilir. Bu sebeplerden dolayı bu tür sistemlerin testi genellikle *gerçek benzeri* ortamlarda yapılır.

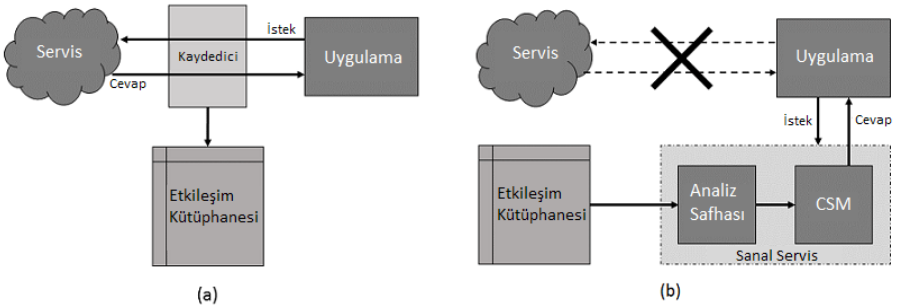
Popüler gerçek benzeri ortam yaratma yollarından birisi *sanal makineleri* kullanıp bağımlı uygulamaların bir kopyasını yaratmaktır [6, 11]. Fakat sanal makinelerin konfigürasyonu ve bakımı bazen maliyetli olabilir. Bunun yanında üçüncü-parti servisler gibi bazı parçaların kopyasını yaratmak mümkün olmayabilir.

Sanal makinelerin bir alternatifi bağımlı servislerin öğrenilip sentetik cevapların üretildiği *servis sanallaştırma*dır. Servis sanallaştırma kavramı sanal sunucular ile karıştırılmamalıdır. Servis sanallaştırma sadece ilgili servisin istek-cevap davranışını simüle etmeyi amaçlar [2]. Bunun yanında servis sanallaştırma konsepti, *mocking*'le de karıştırılabilir. Fakat mock objeleri belli bir geliştirme ihtiyacını gidermek için o anlık üretilmiş basit davranışları taklit eden objelerdir. Buna karşın sanal servisler bütün geliştirme ihtiyaçlarını karşılar, tekrar tekrar kullanılabilir ve üretim zincirinin herhangi bir anında konuşlandırılabilir.

Otomatik sanal servis yaratmanın iki yolu vardır. Birinci metod sanallaştırılacak servisin WSDL gibi servis spesifikasyonlarını veren bir dökümanın elde olduğunu varsayar. Bu yöntem gelen bir isteğe geçerli bir cevap dönüleceğini garanti eder. Fakat bu yöntemin bir takım kısıtları vardır. Örneğin sanallaştırılmak istenen servis üçüncü-parti bir servis olabilir. Bunun yanında bu yöntem zaman açısından maliyetlidir [8].

İkinci ve aynı zamanda bizim kullandığımız yöntem servis ayaktayken, test edilecek yazılım ve servis arasında kaydedilmiş istek-cevap ikililerini kullanır

[13]. Yaklaşımımızda, servis henüz ayaktayken kaydedilmiş istek-cevap ikililerinden servisi öğrenerek yeni gelen bir isteğe cevap sentezlemeye çalışıyoruz. Kullandığımız teknikler mesaj formatından bağımsız çalışır. Yaklaşımımızın temel hatları Şekil 1'de görülebilir.



Şekil 1. FancyMock'a genel bakış. (a) Sanallaştırılacak servisin ulaşılabilir olduğu durum. Servis ve test edilecek olan uygulama arasında duran bir kaydedici istek-cevap ikililerini bir etkileşim kütüphanesine kaydeder. (b) Sanallaştırılacak servisin ulaşılamaz olduğu durum. Uygulama sanal servise bir istek gönderir ve Cevap Sentezleme Motoru (CSM) analiz safhasının ışığında bir cevap üretip gönderir.

Yaklaşımımızın temel katkıları şöyledir:

- Bu çalışmada mesaj formatından bağımsız çalışan otomatik bir servis sanallaştırma yaklaşımı sunduk.
- Kayıtlı istek-cevap ikililerini işlemek için bioinformatik ve makine öğrenmesi algoritmaları kullandık.
- Bu çalışmada sunduğumuz teknik mesaj uzunluğu konusunda herhangi bir sınırlama getirmez ve üretilen cevaplar mesaj formatına uygundur.
- Bütün geliştirmelerimizi FancyMock adını verdiğimiz bir araç olarak gerçekleştirdik ve yaklaşımımızı gerçek servisler üzerinde doğruladık.

Bildirinin geri kalanı şu şekilde düzenlenmiştir: İkinci kısım bu konuda daha önce yapılmış çalışmaları anlatır. Üçüncü kısım yaklaşmımızın detaylarını verir. Dördüncü kısım deneyleri sunarak yaklaşımızı değerlendirir. Son olarak beşinci kısım ise kısa bir özetle bildiriyi sonlandırır.

2 İlgili Çalışmalar

Servis sanallaştırma oldukça yeni bir kavramdır. Bu sebeple, bu konu üzerine yapılmış çok sayıda akademik çalışma bulunmamaktadır. Fakat farklı alanlarda benzer problemlere bazı çözümler önerilmiştir. Örneğin Cui vd. [7] bir uygulamanın network izlerini inceleyerek uygulamanın kullandığı protokol mesaj formatlarını çıkaran bir araç sunmuştur. Bu araç protokolden bağımsız çalışır fakat bu yöntem sanal servislerin yaratılması için uygulanamaz.

Servis temelli ortamların test edilmesinin zorluklarını inceleyen çalışmalar da bulunmaktadır. Morris vd. [14] bu tür mimarilerin test edilmesinin zorluklarını ve bu konuda sanal servislerin nerede durduğunu anlatan geniş bir çalışma yayınlamıştır. Bozkurt vd. [5] ise servis odaklı mimarilerin test edilmesindeki ve entegrasyon testindeki zorlukları anlatan kapsamlı bir derleme makalesi yayımlamıştır. Nizamic vd. [16] gerçek iş hayatında sanal servislerin kullanılmasını gösteren bir vaka çalışması yapmıştır.

Servis sanallaştırma konusuna odaklanan son çalışmalar [8,9,18,20,21]'de sunulmuştur ve yazarlar sanal servis yaratmayı ele alan bir dizi çözüm önermiştir. Bu çalışmalar da bioinformatik algoritmalarından faydalanmaktadır. Bu çalışmalardan sonuncusu daha öncekilerde önerilen çözümleri ilerleterek Opaque Service Virtualization (OSV) adıyla yeni bir araç sunmuştur. OSV kaydet-oynat yapısıyla çalışmaktadır ve mesaj formatından (XML, JSON vs.) bağımsızdır. Fakat OSV'nin başarılı bir şekilde çalışabilmesi için mesajların ya belirli bir uzunlukta olması ya da uzunluk bilgisinin mesaja gömülmüş olması önerilmektedir [1]. Bunun yanında Deneyler kısmında gösterildiği gibi kaydedilen mesajların sayısı arttığında oldukça kötü bir performans göstermektedir.

Sanal makineler ya da Docker [4] gibi yeni çıkan konteyner tarzı araçlar servis sanallaştırmaya bir alternatif değildir; bu kavramlar birbirini tamamlayıcı niteliktedir. Örneğin bir sanal servis bir konteynerin içinde çalıştırılabilir.

3 Yöntem

FancyMock aracının genel şeması Şekil 1'de görülmektedir. Şekil 1(a) servis hala ulaşılabilirken yaşanılan senaryoyu göstermektedir. Bu senaryoda, uygulama servisle etkileşime geçer. Bu etkileşim devam ederken istekler ve ilgili cevaplar etkileşim kütüphanesine bir kaydedici vasıtasıyla kaydedilmektedir. Şekil 1(b) ise servisin herhangi bir sebepten dolayı ulaşılamadığı durumu göstermektedir. Bizim katkımız da bu durumda ortaya çıkmaktadır. İlk olarak Analiz kısmında etkileşim kütüphanesindeki veri analiz edilir ve gerekli bilgiler öğrenilir. Bu bilgiler ışığında Cevap Sentezleme Motoru (CSM) yaratılır. Sanal servise yeni bir istek ulaştığında, Cevap Sentezleme Motoru yeni bir cevap üretir.

Aşağıdaki kısımlarda Analiz motoru ve Cevap Sentezleme Motorunun detayları anlatılmaktadır.

3.1 Analiz Motoru

Bu kısımda ilk olarak her bir istek ve cevap tipi için bir *şablon* oluştururuz. Cevap Sentezleme Motoru bu *şablonları* daha önce kaydedilmemiş bir istek geldiğinde yeni bir cevap sentezlemek için kullanır. İsteğin kayıtlı olduğu durumlarda doğru cevap etkileşim kütüphanesinde kolayca bulunabilir.

Genel olarak, eğer iki istek birbirine benziyorsa bunların cevapları da birbirine benziyor demektir. Fakat buna hata mesajları ya da durumsal servisler gibi iki istisna gösterebiliriz. Hata mesajları bizim önerdiğimiz çözümde göz önünde bulundurulmuştur. Fakat durumsal servisler bu çalışmanın kapsamı dışındadır.

İsteklerin birbirine benzeme durumu ikililer arasındaki uzaklığa bakarak karar verilir. Uzaklık ise biyoinformatikte kullanılan Needlman-Wunsch ikili hizalama algoritmasının [15] çıktısındaki eşleşmeler ve hizalama uzunluğu üzerinden hesaplanır.

Tablo 1. İki farklı tipte (searchsong, searchuser) istekten oluşan yapay bir etkileşim kütüphanesi. Aynı ID'li istekler ve cevaplar birbirlerine karşılık gelmektedir.

ID	İstekler
1	{**type:** searchsong, **title:** somesong, **singer:** somesinger}
2	{**type:** searchuser, **username:** first}
3	{**type:** searchuser, **username:** second}
4	{**type:** searchuser, **username:** third}
5	{**type:** searchuser, **username:** fourth}
6	{**type:** searchsong, **title:** anothersong, **singer:** somesinger}

ID	Cevaplar
1	{**type:** searchsong, **title:** somesong, **streamurl:** someurl}
2	{**type:** searchuser, **fname:** John, **lname:** Doe}
3	{**type:** searchuser, **errormessage:** Not Found}
4	{**type:** searchuser, **errormessage:** Not Found}
5	{**type:** searchuser, **fname:** Jane, **lname:** Roe}
6	{**type:** searchsong, **title:** anothersong, **streamurl:** anotherurl}

Etkileşim kütüphanesinde kayıtlı olmayan bir isteğe en yakın isteği bulmanın en sade yolu yeni gelen isteğin bütün isteklerle olan uzaklığını hesaplamaktır. Fakat, bu kontrolü her seferinde gerçekleştirmek özellikle büyük etkileşim kütüphaneleri için büyük bir vakit kaybı olabilir.

Bunun yerine birbirine benzer kayıtlı mesajları orijinal kNN [17] algoritmasının bizim tarafımızdan modifiye edilmiş versiyonunu kullanarak bir araya getirme yolunu izledik. Orijinal kNN algoritması öbekleme için kullanılan bir makine öğrenmesi algoritmasıdır. Modifiye edilmiş kNN Algoritma 1'de gösterilmiştir. Modifiye edilmiş kNN algoritmasına göre belli bir uzaklığın içindeki mesajlar aynı kümeye konulur. Eğer ikililerin aralarındaki uzaklık belli bir eşik değerinin üzerindeyse yeni bir küme oluşturulur. Bu eşik değeri deneylerden yola çıkarak 0.8 olarak belirlenmiştir. Hatasız bir durumda her bir kümenin içinde sadece bir tek tip istek veya cevap bulunur ve tipleri aynı olan iki istek farklı kümelerde bulunmaz. Örneğin bütün *searchsong* tipli istekler aynı kümede bulunur ve bu kümenin dışında bir *searchsong* bulunmaz. Kümeleme işlemi yapıldıktan sonra her bir küme/tip için bir şablon çıkarılır. Böylece sanal servise yeni bir istek geldiğinde eğer o istek daha önce kaydedilmemiş ise, ona en yakın mesajları bulmak için sadece şablonlarla karşılaştırmak yeterli olacaktır. Kümeleme işlemi hem istekler hem de cevaplar için ayrı ayrı yapılır. Tablo 1'de veriler için muhtemel kümeler aşağıdaki gibi olur:

İstek Kümeleri: $\{\mu_1{:}\{1, 6\}, \mu_2{:}\{2, 3, 4, 5\}\}$
Cevap Kümeleri: $\{\nu_1{:}\{1, 6\}, \nu_2{:}\{2, 5\}, \nu_3{:}\{3, 4\}\}$

Algorithm 1 Modifiye Edilmiş K-Nearest Neighbour

Input:

$\Re$: İstek veya cevapların bulunduğu bir liste.
τ : Küme ayırma eşiği.
K : K parametresi.

Output:

C : Anahtarları istek veya cevaplar, değerleri küme numaraları olan bir sözlük.

```
kümeNumarası ← 0
C[ℜ[0]] ← kümeNumarası                ▷ Birinci elemana küme numarası atanır.
kümelenmişMesajlar.ekle(ℜ[0])
for each r in ℜ do
    uzaklıklar, numaralar ← KNearestNeighbourBul(K,r,kümelenmişMesajlar)
    if min(uzaklıklar)> τ then
        kümeNumarası++                ▷ Yeni bir küme yarat.
        C[r] ←kümeNumarası
    else
        C[r] ← enÇokBulunanElemanıAl(numaralar) ▷ En yakın K tane arasında
    en çok bulunan küme numarası.
    end if
    kümelenmişMesajlar.ekle(r)
end for
```

Kümeleme işlemi bittikten sonra istek kümeleri ve cevap kümeleri arasındaki ilişkinin bulunması gerekir. Bunun sebebi aynı istek kümesinde olan mesajların cevaplarının farklı kümelerde olabilme ihtimalidir. Örneğin 2 ve 3 numaralı istekler, μ_2 kümesinde bulunurken, cevaplar sırasıyla ν_2 ve ν_3 kümelerinde bulunur. Cevap Sentezleme Motoru cevapları üretirken daha gerçekçi cevaplar için bu ilişkiyi göz önünde bulundurur. Bu işleme *küme eşleştirmesi* adı verilmiştir.

Pre-sampling Kümeleme işleminin bize hız kazandırdığını ve sadece bir kere yapılması gerektiğini daha önce anlatmıştık. Fakat kümeleme işlemi de her ikili arasındaki uzaklığı bulmak gerektiğinden oldukça maliyetli bir iştir. Bu çalışmada, performansı artırmak için, basit ama etkili bir çözüm olarak kümelerin eleman sayısına bir üst sınır koyduk ve bu işleme pre-sampling adını verdik. Pre-sampling yöntemiyle öncekiyle aynı sayıda küme olmasına rağmen ikililerin karşılaştırma sayısı azaldığı için kümeleme işlemi hız kazanacaktır. Çünkü kümeleme işleminin karmaşıklığı kareselden doğrusala düşer. Örneğin, eğer küme üst sınırını 3 seçerek pre-sampling uygulamış olsaydık, μ_2 şunun gibi olurdu: μ_2:{2, 3, 4}

Pre-sampling'in cevap üretimine olan etkisini Deneyler kısmında değerlendireceğiz.

Şablon Çıkarma Bu aşamada bir çoklu dizi hizalama algoritması olan ClustalW'yi [19] kullanarak her bir kümeyi temsil eden bir şablon çıkaracağız. Şablon çıkarma işlemi hem istek hem de cevap kümeleri için yapılır.

Çoklu dizi hizalama işlemi yapıldıktan sonra, aynı indisteki karakterler arasında bir *uyum* yoksa şablonun o indisine bir joker karakter konulur. *Uyum* o indisteki karakterlerin %80'inin aynı olması olarak belirlenmiştir. Eğer bir uyum varsa

şablonun o indisine *uyum*u sağlayan karakter konulur. Bu tanıma göre istek kümesi μ_2'nun ve cevap kümesi ν_1'in şablonu aşağıdaki gibi olur:

şablon μ_2: {`type:searchuser, username:#######`}
şablon ν_1: {`type:searchsong, title:########, streamurl:#######`}

Bu yaklaşım dizilerin asıl kısımlarını tutarken değişken kısımlarını joker karakterlerle (#) değiştirir.

Aşağıdaki iki şablon geçersiz şablonlara örnektir. Birinci şablonda değişken olan bir kısım tamamiyle joker karakterlerden oluşmamaktadır. İkinci şablonun problemi ise değişken olmayan bir kısmın joker karakterleri barındırmasıdır.

```
{type:searchsong, title:####s###, streamurl:#######}
{type:sear##song, title:########, streamurl:#######}
```

Hatırlanacağı üzere, şablonlar, bir isteğin hangi kümeden olduğuna karar vermek için kümelerdeki bütün elemanlar yerine sadece bir karşılaştırma yapmak için üretilirler.

Bu çalışmada şablon çıkarılırken [21]'de anlatılan yöntemden esinlenilmiştir.

Mesajlardaki Değişken Kısımların Analizi Önceki çalışmalar gerçekçi cevaplar üretmek konusunda kısıtlı yeteneklere sahiptir [9, 21]. Biz bu çalışmada, çeşitli ve gerçekçi cevaplar üretebilmeyi amaçlıyoruz. Bu adımda kayıtlı bütün cevap mesajlarının bütün değişken kısımlarını bulmayı hedefliyoruz. Değişken kısımları yani mesajların parametrelerini bulmak için mesajı o kümenin şablonu ile hizalarız. Bu hizalamada şablonun joker karakterleriyle eşleşen kısımlar o mesajın değişken kısımları olur. Bu kısımlar çıkarıldıktan sonra bir *sözlük* yapısında tutulur. Aşağıda bir örnek gösterilmiştir.

```
{type:searchsong, title:somesong, streamurl:someurl}
{type:searchsong, title:########, streamurl:#######}
```

Yukarıdaki hizalamaya bakarak *somesong* ve *someurl* in bir değişken kısım olduğunu kolayca anlayabiliriz. Bu işlem bütün cevap kümelerinin bütün mesajlarına uygulanır ve *değişkenKısımSözlüğü* ismiyle bir sözlük yaratılır. Örnek bir *değişkenKısımSözlüğü* Tablo 2'de gösterilmiştir.

Tablo 2. Tablo 1 için *değişkenKısımSözlüğü*nün anahtar ve değerleri gösterilmiştir. Örneğin (2,1) ikinci kümenin birinci değişken kısmı anlamına gelmektedir.

Key	Value
(1,1)	[somesong, anothersong, ...]
(1,2)	[someurl, anotherurl, ...]
(2,1)	[John, Jane, ...]
(2,2)	[Doe, Roe, ...]

3.2 Cevap Sentezleme Motoru

Cevap Sentezleme Motoru (CSM), Analiz kısmının çıktılarını kullanarak kabul edilebilir, gerçeğe yakın ve çeşitlendirilmiş cevaplar üretir. Analiz kısmının çıktıları şablonlar, kümeler ve *değişkenKısımSözlüğü*dür.

CSM, servise yeni bir istek geldiğinde ona cevap sentezlemek için harekete geçer ve toplamda 3 ana adımdan oluşur:

1. Cevap sentezlemede temel alınacak mesajın seçilmesi. (Temel Mesaj Seçimi):
   (a) En benzer istek kümesinin şablonunu bul.
   (b) Seçilen istek kümesinin ilişkili olduğu cevap kümelerini bul.
   (c) Seçilen cevap kümesinden bir mesaj seç (*Temel cevap*) ve bu cevabın isteğini bul (*Temel istek*).
2. Simetrik Kısımların Yerleştirilmesi (SKY) prosedürünü uygula:
   (a) *Temel istek* ve *Temel cevap* arasında aynı olan *değişken kısımları*n indislerini bul.
   (b) Bu indislerden yola çıkarak *Gelen isteğin* ilgili alanlarını *Temel cevaba* kopyala.
3. *Temel cevabın* geri kalan değişken alanlarını *değişkenKısımSözlüğü* yardımıyla değiştir.(Çeşitlendirme).

Temel Mesaj Seçimi Öncelikle, yeni gelen bir isteği bütün istek kümelerinin şablonlarıyla karşılaştırır ve en yakınını buluruz. Aynı tipteki istekler farklı tiplerdeki cevaplarla yanıtlanabildiğinden muhtemel cevap kümelerinden birini seçebiliriz. Burada en yüksek ihtimalli cevap kümesini seçmek uygun olur. Daha sonra seçilen bu cevap kümesinden rastgele bir mesaj alınır (*Temel cevap*) ve buna karşılık gelen istek bulunur (*Temel istek*).

Simetrik Kısımların Yerleştirilmesi (SKY) Bu adımda *Temel istek* ve *Temel cevap* arasında tamamen aynı olan değişken kısımlar bulunur. Aşağıdaki örnekte görüldüğü gibi *somesong* bu ikili arasında aynı olan bir değişken kısımdır. Çalışmamızda sadece değişken kısımlar arasındaki simetri hesaba katılmıştır. Örneğin *searchsong* aşağıdaki ikili arasında simetrik olarak kabul edilmemiştir. Çünkü bu alan şablonun kendisinde bulunmaktadır (kümedeki bütün mesajlarda ortaktır).

Temel istek: {`type:searchsong, title:somesong, singer:somesinger`}
Temel cevap: {`type:searchsong, title:somesong, streamurl:someurl`}

Bu alanlar bulunduktan sonra *Temel cevap* üzerinde değişiklikler yapılarak yeni gelen isteğe karşılık gelen cevap sentezlenmeye başlanır. [21]'deki çalışmanın aksine biz sadece değişken kısımlar arasındaki simetriyi hesaba katarız, şablon üzerindeki simetriyi göz ardı ederiz. Bunun yanında yeni yerleştirilen değişken kısım *Temel cevabın* ilgili kısmından daha kısaysa o kısım küçülür, eğer daha uzunsa büyür. Bu bize değişen uzunluktaki mesajların üstesinden gelmemizi sağlar. Eğer mesajlar arasında simetrik bir değişken kısım yoksa bu adım atlanır.

Çeşitlendirme *Temel cevabın* simetrik olmayan değişken kısımları *değişkenKısımSözlüğü*nün ilgili yerinden seçilerek değiştirilir. Bu şekilde, sentezlenen cevaplar çeşitlendirilir ve bu, yazılımın daha kapsamlı şekilde test edilmesine yardımcı olur. Aşağıda örnek bir *Çeşitlendirme* işlemi gösterilmiştir.

Yeni gelen istek 1 ve Sentezlenen Cevap 1:
{`type:searchsong, title:awesomesong, singer:coolsinger`}
{`type:searchsong, title:awesomesong, streamurl:randomurl1`}

4 Deneyler

Bu kısımda FancyMock'u değerlendirmek için yaptığımız deneyleri sunacağız. Bu çalışmada FancyMock'u Baseline adını verdiğimiz bir referans implementasyonla karşılaştırdık. Baseline implementasyonunda pre-sampling, değişken kısım analizi ve çeşitlendirme adımları yoktur. SKY prosedürünün ilkel bir versiyonu dahil edilmiştir. Bu haliyle Baseline [21]'de anlatılan tekniğin (OSV) bizim tarafımızdan kodlanmış versiyonudur.

Yaklaşımımızı değerlendirmek için deneylerde 4 ölçüm yaptık: sentezlenen cevapların geçerliliği, sentezlenen cevaplar arasındaki ortalama uzaklık (çeşitlilik), analiz safhası çalışma süresi ve ortalama cevap sentezleme süresi. Bunun yanında pre-samplingin şablon çıkarma üzerinde olan etkisini de değerlendirdik.

Deneyler 32 GB hafızalı ve Intel Xeon E5520 2.27 GHz işlemcili bir sunucuda koşulmuştur.

4.1 Veri setleri

Yaklaşımımızı üç farklı veri seti üzerinde test ettik. Daha gerçekçi sonuçlar için bütün veriler gerçek sistemlerden toplanmıştır. Veriler bilgisayar ortamından otomatik olarak sorgu atan küçük kod parçalarıyla toplanmıştır. Birinci veri seti İkamet Bilgisi Sorgu Sistemi (İBSS) adı verilen bir sistemden toplanmıştır. Bu sistemde kişi bir id veya tam isimle sorgu yapabilir. Bu isteklere dönen cevaplar ise sorgulanan kişinin doğum tarihi, doğum yeri adres vs. gibi detaylı bilgilerini içermektedir. İkinci veri seti SoundCloud uygulamasının API'sinden toplanmıştır. SoundCloud herkese açık olarak *RESTful* bir API sunmaktadır. Son olarak, WeatherUnderground API'sinden geçmişteki bir güne ait bir hava durumu verisi topladık. Bu API de SoundCloud gibi *RESTful* bir API'dir. Tablo 3 istek tiplerinin ve toplanan mesajların toplam sayısını göstermektedir.

Tablo 3. Veri Setleri

İsim	Format	#Request Type	#Traces
İBSS	XML	4	1200
SoundCloud API	JSON	3	1200
WeatherUnderground API	JSON	1	1200

Her bir veri setini topladıktan sonra karıştırdık ve %60'ını eğitmek için geri kalanını da test için kullandık.

4.2 Geçerlilik Değerlendirmesi

Bu deneyde her bir sentezlenen cevabın geçerli olup olmadığı kontrol edilmiştir. Bir mesaj iki durumda geçersiz olabilir: Ya sentezlenen cevap beklenen mesaj formatına uygun değildir (*JSON, XML, etc.*) ya da beklenen cevap tipine ait değildir. Örneğin *searcsong* tipli cevap beklerken *searchuser* tipli cevap üretilmesi gibi. Geçerlilik deneyinde sadece kayıtlı olmayan (eğitim setinde olmayan) mesajlar kullanılmıştır.

Tablo 4. Sentezlenen cevapların geçerlilik yüzdesi.

İsim	Baseline	FancyMock
İBSS	48.6%	94.8%
SoundCloud API	18.3%	76.1%
WeathrUndrgrnd API	21.2%	100%

Tablo 5. Bütün sentezlenen cevap çiftlerinin arasındaki ortalama uzaklık.

İsim	Baseline	FancyMock
İBSS	0.21	0.71
SoundCloud API	0.09	0.66
WeathrUndrgrnd API	0.11	0.43

Tablo 4 geçerlilik deneylerinin sonuçlarını sunmaktadır. Bu tablo göstermektedir ki, FancyMock mesaj formatından bağımsız olarak geçerli cevaplar üretebilmektedir.

4.3 Çeşitlilik Değerlendirmesi

Bu deneyde çeşitliliği üretilen cevaplar arasındaki farklılık olarak tanımladık. Çeşitliliğin ölçümünü ise üretilen cevaplardan aynı tipte olan her ikili arasındaki uzaklığı hesaplayarak yaptık. Örneğin tipi *searchuser* olan bütün cevapların aralarındaki uzaklığı ikili ikili hesapladık. Daha sonra bu uzaklıkların ortalamasını aldık.

Tablo 5 Baseline yaklaşımı tarafından üretilen cevapların sınırlı bir çeşitliliği olduğunu gösteriyor. Diğer taraftan, FancyMock birbirinden farklı cevaplar üretebilmektedir.

4.4 Analiz Safhasının Çalışma Süresi Değerlendirmesi

Bu deneyle analiz safhasının çalışma süresini değerlendirdik. FancyMock'u, pre-sampling adımını içermeyen Baseline yaklaşımıyla karşılaştırdık.

Tablo 6. Analiz Safhası Çalışma Süresi (hh:mm:ss)

	Baseline			FancyMock		
	400	800	1200	400	800	1200
İBSS	01:53:03	08:47:17	20:11:08	01:22:07	02:37:08	03:54:12
SoundCloud API	00:08:47	00:16:03	00:41:06	00:18:23	00:35:12	00:52:34
WeathrUndrgrnd API	00:10:30	00:39:43	01:24:34	00:16:04	00:32:58	00:52:12

Bu deneyi performanstaki değişimi görebilmek için farklı büyüklüklerdeki (400, 800, 1200) verilerle gerçekleştirdik. Tablo 6 sonuçları göstermektedir. Tablodan anlaşıldığı üzere pre-sampling tekniği özellikle etkileşim kütühanesi büyüdüğünde faydalı olmaktadır. Gerçek hayatta büyük etkileşim kütüphaneleri daha sık karşılaşılan bir senaryodur. Analiz safhasının çalışma süresini etkileyen bir diğer faktör etkileşim kütüphanesindeki mesajların ortalama uzunluğudur. SoundCloud veri setinde Baseline yaklaşımının FancyMock'tan daha iyi performans göstermesinin sebebi mesajların oldukça kısa olmasıdır. FancyMock mesajlar uzadıkça daha büyük bir fark yaratır.

4.5 Ortalama Cevap Sentezleme Süresi

Bu deneyde, cevap sentezleme süresi, servise bir isteğin ulaşmasıyla, bu servisin bir cevap döndürmesi arasında geçen zaman olarak tanımşanmıştır. Tablo 7 ortalama cevap üretim sürelerini göstermektedir. Bu deneyde gerçek servisten topladığımız cevap dönme sürelerini de karşılaştırdık. Tabloya göre bizim önerdiğimiz yaklaşımla bir sanal servisten cevap almak genelde daha uzun sürmektedir. Bunun sebebi üretilen mesajların geçerlilik oranını ve çeşitliliğini artırmak için Cevap Sentezleme Motoruna eklediğimiz adımlardır. WeatherUnderground API'ında istisnai bir durum olarak FancyMock daha iyi performans göstermiştir. Bunun sebebi istekler ve cevaplar arasında simetrik değişken kısımlar bulunmadığı için SKY adımının FancyMock'ta atlanmış olmasıdır. Baseline yaklaşımında ise mesajın değişken olmayan kısımları arasında da yerleştirme yapılmaktadır.

Tablo 7. Ortalama Cevap Üretim Süresi (saniye)

	Baseline			FancyMock			Gerçek Servis		
	Min	Max	Ort	Min	Max	Ort	Min	Max	Ort
İBSS	1.42	1.91	1.68	2.50	3.92	3.09	0.02	8.94	0.61
SoundCloud API	0.47	0.91	0.72	0.33	1.22	0.86	0.10	0.54	0.45
WeathrUndrgrnd API	0.69	0.84	0.74	0.57	1.31	0.66	0.05	0.96	0.63

4.6 Pre-sampling'in Şablon Çıkarma Üzerine Etkisi

Son olarak, pre-sampling tekniğinin şablon çıkarma üzerine olan etkisini inceleyeceğiz. Bir şablonda mesajın değişken olmayan kısımlarının korunması ve bütün değişken kısımların joker karakterleriyle doldurulmuş olması beklenmektedir. Doğru ve bozuk şablon örnekleri Şablon Çıkarma başlığı altında görülebilir.

Bu ölçüm veri setinden ve mesajların istek ya da cevap olmasından bağımsızdır. Bu sebeple bu deneyde İBSS veriseti kullanılmıştır. Küme üst sınırı olarak 5, 10, 20, 25 ve sınırsız (Baseline yaklaşımındaki gibi) seçtik ve daha doğru bir sonuç için bu deneyi 10 kere tekrarlayıp ortalamasını aldık.

Tablo 8 deneyin sonuçlarını göstermektedir. Tablodan görüldüğü üzere küme üst sınırı arttıkça şablonun doğruluğu artmaktadır. Fakat belli bir yerde doğruluk oranı doyum noktasına ulaşmaktadır. Buna göre eğer bütün kümeyi almak yerine pre-sampling yaparak küme üst sınırını uygun bir büyüklükte seçersek, ikili uzaklık bulma işlemi azalacağından hız konusunda belirli bir artış elde edebiliriz.

Tablo 8. Pre-sampling'in Şablon Çıkarma Üzerine Etkisi

Küme Üst Sınırı	Şablon Geçerlilik Oranı (%)
5	82.5
10	92.5
20	95.0
25	95.0
Bütün Küme	95.0

4.7 Kısıtlar

Bu çalışmayı yaparken yaklaşımımızın belli kısıtları olduğunu gördük. Bizim kullandığımız modifiye edilmiş kNN tekniği ancak uygun bir eşik değeri seçilirse doğru çalışabilir. Bunun yerine DBSCAN [10] ya da OPTICS [3] daha gelişmiş kümeleme teknikleri kullanılabilir. Bunun dışında FancyMock örneğin bir saniyenin altında gibi çok kısa bir sürede cevap bekleyen uygulamalar ve şifreli mesajlar kullanan protokoller için uygun değildir.

5 Sonuç

Bu çalışmada özgün bir servis sanallaştırma tekniği geliştirdik. Bizim yaklaşımımızda servis henüz ayaktayken istekler ve cevaplar kaydedilir. Daha sonra belirli tekniklerle bir şablon çıkarılır ve sonuç olarak yeni gelen bir istek için sentetik bir cevap yaratılır.

FancyMock mesaj formatından bağımsız çalışır yani sanallaştırılacak servisin belirli bir mesaj formatında çalıştığını varsaymaz. İkinci olarak, üretilen sentetik cevaplar hem mesaj formatını ihlal etmez hem de doğru tiptedir. Bu sebeple üretilen cevapların geçerlilik oranı yüksektir. Üçüncü olarak üretilen cevaplar olabildiğince çeşitlendirilerek yapmacıklıktan olabildiğince uzaklaşılmıştır ve son olarak FancyMock'un performansı pratikte kabul edilebilir sınırlar içerisindedir.

FancyMock'la yaptığımız deneyler yaklaşımımızın faydalarını göstermiştir. Bir sonraki çalışma olarak durumsal servislerin sanallaştırılması üzerine yoğunlaşmayı planlıyoruz.

Kaynaklar

1. ODP manual page. https://docops.ca.com/devtest-solutions/9-5/en/using/using-ca-service-virtualization/using-devtest-workstation-with-ca-service-virtualization/creating-service-images/create-a-service-image-by-recording/transport-protocols/opaque-data-processing-transport-protocol, accessed: 2017-08-26
2. Smartbear (2017), https://smartbear.com/learn/software-testing/what-is-service-virtualization/, [Online; accessed 23-May-2017]
3. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: ordering points to identify the clustering structure. In: ACM Sigmod record. vol. 28, pp. 49–60. ACM (1999)
4. Boettiger, C.: An introduction to docker for reproducible research. ACM SIGOPS Operating Systems Review 49(1), 71–79 (2015)

5. Bozkurt, M., Harman, M., Hassoun, Y.: Testing and verification in service-oriented architecture: a survey. Software Testing, Verification and Reliability 23(4), 261–313 (2013)
6. Chen, P.M., Noble, B.D.: When virtual is better than real [operating system relocation to virtual machines]. In: Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on. pp. 133–138. IEEE (2001)
7. Cui, W., Kannan, J., Wang, H.J.: Discoverer: Automatic protocol reverse engineering from network traces. In: Usenix Security. vol. 158 (2007)
8. Du, M., Schneider, J.G., Hine, C., Grundy, J., Versteeg, S.: Generating service models by trace subsequence substitution. In: Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures. pp. 123–132. ACM (2013)
9. Du, M., Versteeg, S., Schneider, J.G., Han, J., Grundy, J.: Interaction traces mining for efficient system responses generation. ACM SIGSOFT Software Engineering Notes 40(1), 1–8 (2015)
10. Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. vol. 96, pp. 226–231 (1996)
11. Hine, C.: Emulating enterprise software environments. Ph.D. thesis, Swinburne University of Technology (2012)
12. Hurwitz, K.: Service Virtualization for dummies. John Wiley & Sons (2013)
13. Michelsen, J., English, J.: What is service virtualization? In: Service Virtualization, pp. 27–35. Springer (2012)
14. Morris, E.J., Anderson, W.B., Balasubramanian, S., Carney, D.J., Morley, J., Place, P.R., Simanta, S.: Testing in service-oriented environments. Tech. rep., Software Engineering Institute, Carneige Mellon University (2010)
15. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of molecular biology 48(3), 443–453 (1970)
16. Nizamic, F., Groenboom, R., Lazovik, A.: Testing for highly distributed service-oriented systems using virtual environments. Proceedings of 17th Dutch Testing Day (2011)
17. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: ACM Sigmod record. vol. 24, pp. 71–79. ACM (1995)
18. Schneider, J.G., Mandile, P., Versteeg, S.: Generalized suffix tree based multiple sequence alignment for service virtualization. In: Software Engineering Conference (ASWEC), 2015 24th Australasian. pp. 48–57. IEEE (2015)
19. Thompson, J.D., Higgins, D.G., Gibson, T.J.: Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic acids research 22(22), 4673–4680 (1994)
20. Versteeg, S., Du, M., Bird, J., Schneider, J.G., Grundy, J., Han, J.: Enhanced playback of automated service emulation models using entropy analysis. In: Continuous Software Evolution and Delivery (CSED), IEEE/ACM International Workshop on. pp. 49–55. IEEE (2016)
21. Versteeg, S., Du, M., Schneider, J.G., Grundy, J., Han, J., Goyal, M.: Opaque service virtualisation: a practical tool for emulating endpoint systems. In: Proceedings of the 38th International Conference on Software Engineering Companion. pp. 202–211. ACM (2016)