T.C.

MALTEPE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Yüksek Düzeyli Mimari Bazlı Simülasyonlarda Dinamik Yük Dengeleme

Dynamic Load Balancing on High Level Architecture Based Simulations

Erdem REŞBER

Yüksek Lisans Tezi

Tez Danışmanı

Prof. Dr. Müslim BOZYİĞİT

İSTANBUL - 2014

T.C.

MALTEPE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Yüksek Düzeyli Mimari Bazlı Simülasyonlarda Dinamik Yük Dengeleme

Dynamic Load Balancing on High Level Architecture Based Simulations

YÜKSEK LİSANS TEZİ

ERDEM REŞBER

Tez Danışmanı

Prof. Dr. Müslim BOZYİĞİT

İSTANBUL - 2014

Bu tez çalışması, Maltepe Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 27/06/2014 tarih ve 2014/14 sayılı kararıyla oluşturulan jüri tarafından Bilgisayar Mühendisliği Yüksek Lisansı Tezi olarak kabul edilmiştir.

JÜRİ

Prof. Dr. Müslim BOZYİĞİT

(Danışman)

Yrd. Doç. Şenol Z. ERDOĞAN

(Üye)

Yrd. Doç. Dr. Fatih YÜCALAR

(Üye)

Erdem REŞBER, ***Yüksek Düzeyli Mimari Bazlı Simülasyonlar Üzerinde Dinamik Yük Dengeleme,*** **İstanbul 2014**

ÖZET

Bilgisayar simülasyonları günümüzde teorik ya da fiziksel gerçek bir sistemin modellenerek, sistemin özelliklerini ve davranışlarını bilgisayar ortamında değerlendirmek ve analiz etmek üzere birçok alanda kullanılmaktadır. Geliştirilen modeller ile işletilen simülasyon yazılımları, karmaşık hesaplama yöntemleri içermesi ve en kısa zamanda ihtiyaç duyulan sonuçları üretebilmeleri amacıyla, yüksek hesaplama kaynaklarına ihtiyaç duymaktadırlar. Yüksek hesap kapasitesine sahip tek işlemcilerin maliyetleri dolayısıyla mevcut çoklu işlemli kaynaklarının etkin bir biçimde kullanılması seçeneği karşımıza çıkmaktadır.

Bu durumda yüksek hesaplama kaynağı gereksinimlerini karşılamak üzere dağıtık yapı yaklaşımı kullanılmaktadır. Bu yaklaşım ile bir ağda bulunan tüm bilgisayar kaynaklarının bir arada kullanılarak tek bir bilgisayar gibi programların çalıştırılması yeteneği kazandırılmıştır.

Dağıtık sistemlerde yüksek olasılıkla birtakım bilgisayarlar aşırı yüklenmişken diğerleri boş veya az yüklenmiş olabilir. Bunun önüne geçebilmek için yük dengeleme algoritmaları kullanılmaktadır.

Bu çalışma kapsamında, HLA (High Level Architecture) bazlı çalışmalar incelenmiş ve kullanılan yöntemler ışığında, yeni bir dinamik yük dengeleme yaklaşımı geliştirilmiştir.

Örnek uygulama olarak, RTI (Run Time Infrastructure) üzerinde, dağıtık kablosuz sensör ağ simülasyonu seçilmiştir. Kullanılan yük dengeleme algoritması durum aktarmalı federe taşınmasına dayanmaktadır. Elde edilen sonuçlardan, yük dengeleme ile belli şartlarda, alternatiflerine göre daha iyi performans elde edilebileceği görülmüştür.

Anahtar Kelimeler: HLA, RTI, Dinamik Yük Dengeleme, Durum Aktarımı, Federe Taşınması

Erdem REŞBER, ***Dynamic Load Balancing on High Level Architecture Based Simulations,*** **İstanbul 2014**

ABSTRACT

Computer simulations are used in many fields today, in order to evaluate and analyze properties and behavior of the system by modelling a theoretical or a real physical system on the computer environment. The simulation applications using complex calculation methods need high computing power to produce better results. Since individual processors have limited capacities and those of high computing capacity are costly, effective use of multiple processor option becomes viable.

In this case, the distributed programming architecture approach is used to meet the needs of high computing power. With this approach, it is possible to use combination of multiple computing resources, in the form of a network or a distributed system.

In distributed environment, it is possible for some computers to be overloaded, while others under loaded. To remedy this problem, load balancing is needed.

Within the scope of this study, the work on HLA based dynamic load balancing have been studied, consequently a new dynamic load balancing approach has been proposed.

As an example application, on RTI (Run Time Infrastructure), a wireless sensor network simulation is chosen. The load balancing algorithm utilises migration of federate state, in case of imbalance. The results show that the load balancing under certain conditions can provide higher performance compared to alternative.

Keywords: HLA(High Level Architecture), RTI, Dynamic Load Balancing, State Migration, Federate Migration

TEŞEKKÜR

Bu tez çalışmasının planlanmasında, araştırılmasında, yürütülmesinde ve oluşumunda ilgi ve desteğini esirgemeyen, engin bilgi ve tecrübelerinden yararlandığım, yönlendirme ve bilgilendirmeleriyle çalışmamı bilimsel temeller ışığında şekillendiren sayın hocam Prof. Dr. Müslim BOZYİĞİT'e sonsuz teşekkürlerimi sunarım.

Tez konusunun seçiminde yardımcı olan ve konuyla ilgili bilgi birikimini paylaşan sayın Dr. Mehmet Fatih HOCAOĞLU'na teşekkür ederim.

Sevgili aileme manevi hiçbir yardımı esirgemeden yanımda oldukları için tüm kalbimle teşekkür ederim.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR

Kısaltma	İngilizcesi	Türkçesi
HLA	High Level Architecture	Yüksek Düzeyli Mimari
DoD	Department of Defense	Savunma Departmanı
DMSO	Defense Modeling and Simulation Office	Savunma Modelleme ve Simülasyon Ofisi
OMT	Object Model Template	Nesne Modeli Taslağı
FOM	Federation Object Model	Federasyon Nesne Modeli
SOM	Simulation Object Model	Simülasyon Nesne Modeli
RTI	Run-time Infrastructure	Koşum zamanı Altyapısı
CPU	Central Processing Unit	Merkezi İşlemci Birimi
LAN	Local Area Network	Yerel Alan Ağı
DAFT	Federate Migration with State Transfer	Durum Aktarımlı Federe Taşınması
IEEE	The Institute of Electrical and Electronics Engineers	Elektrik Elektronik Mühendisleri Enstitüsü
API	Application Programming Interface	Uygulama Programlama Arayüzü
TCP	Transmission Control Protocol	İletim Kontrol Protokolü
IP	Internet Protocol	İnternet Protokolü
MVC	Model View Controller	Model Görünüm Kontrolü

ŞEKİLLER DİZİNİ

1 GİRİŞ

Bilgisayar simülasyonları yüksek hesaplama yoğunluğuna sahip yazılımlardır. Yüksek hesaplama kaynağı elde etmenin bir yolu da dağıtık hesaplama kullanmaktır. Dağıtık sistemler, teorik olarak, kullanıcılarına tek bir bilgisayar gibi görünen bağımsız bilgisayarların bileşimidir. Bir dağıtık sistemde yüksek olasılıkla birtakım bilgisayarlar aşırı yüklenmişken diğerleri boş veya az yüklenmiş olabilir; bu durum performans kayıplarına neden olur. Bunun önüne geçebilmek için yük dengeleme algoritmaları kullanılır. Yük dengeleme algoritmalarında en sık kullanılan yöntemler; yeni bir prosesin yaratılması sırasında bunu daha az yük yoğunluğuna sahip bir düğümde yaratmak, ya da o an çalışan prosesin yüksek yük yoğunluğuna sahip düğümden, daha düşük yük yoğunluğuna sahip düğüme aktarımını yapmaktır.

Günümüzün dağıtık bilgisayar simülasyon standartlarından bir tanesi Yüksek Seviyeli Mimari (High Level Architecture - HLA) ortamıdır. HLA, simülasyon bileşenlerinin birlikte çalışabilmeleri ve yeniden kullanılabilmeleri için U.S. DMSO tarafından geliştirilmiş ve bir Elektrik Elektronik Mühendisleri Enstitüsü (The Institute of Electrical and Electronics Engineers - IEEE) standardı olarak kabul edilmiştir. HLA kapsamında, bir simülasyon federasyon olarak ve bir simülasyon bileşeni ise federe olarak adlandırılır. Federeler doğrudan birbirleriyle iletişim halinde değillerdir ve tüm iletişim Koşum Zamanı Altyapısı (Run Time Infrastructure - RTI) ile gerçekleştirilmektedir. RTI servisleri, federasyon koşumu sırasında veri değiş tokuşunu ve federeler üzerinde yapılan eşzamanlı işlemleri gerçekleştirmektedir [1][2][3][4][5].

RTI, ek fonksiyonellikler kazandırılması bakımından özgürdür. Ancak günümüzde RTI uygulamaları kendi bünyesinde hala herhangi bir yük dengeleme mekanizması barındırmamaktadır, bu yüzden performans kayıplarına neden olan yük dengesizlikleri meydana gelebilmektedir. HLA simülasyonları, bu yüzden yük yönetim mekanizmalarına ihtiyaç duymaktadır.

Tez kapsamında, dağıtık HLA bazlı simülasyonlar üzerinde dinamik yük dengeleme işlemi, federe aktarımı (migration) yöntemi ile gerçekleştirilmiştir. Dinamik yük dengeleme işlemi için ağ üzerindeki düğümlerin (bilgisayarların) yükleri belirli zaman aralıklarında izlenmiş ve bir karar verme mekanizması içerisinde gerekli yük dengeleme politikası olan federe aktarım işlemi gerçekleştirilmiştir. Bu çalışmada HLA RTI uygulaması olan Portico kullanılmıştır. Portico Java tabanlı geliştirilmiş ve C++ arayüzleri ile C++ uygulamalarında da kullanılabilmesi sağlanmıştır [6][7].

Tez kapsamında sırasıyla 1. bölümde yapılacak çalışma hakkında giriş bilgisi ve dağıtık sistemler hakkında temel bilgiler verilmiştir. Bunun yanı sıra HLA RTI simülasyonları hakkında bilgiler verilmiştir. 2. bölümde daha önce kullanılmış olan federe göç tekniklerinden bahsedilmiş ve tez konusu üzerinde neler yapılacağı anlatılmıştır. 3. bölümde HLA simülasyonu üzerinde federe aktarım protokolü verilmiştir. 4. bölümde tasarlanan yük dengeleme mekanizması verilmiş ve karar düğümleri anlatılmıştır. 5. bölümde geliştirilen yazılım mimari yaklaşım verilmiştir. 6. bölümde geliştirilmiş olan uygulama üzerinde testler yapılmış ve elde edilen performans sonuçları değerlendirilmiştir ve 7. bölümde sonuçlar toparlanmış, çıkarımlar yapılmış ve gelecekteki muhtemel geliştirmeler verilmiştir.

1.1 Dağıtık Bilgisayar Sistemleri

Tanenbaum'un tanımıyla, dağıtık bir sistem bağımsız bilgisayarlardan oluştuğu halde kullanıcıya tek bir bilgisayar gibi görünen tümleşik bir sistemdir. Bu tanımdan iki temel özellik göze çarpmaktadır. Bunlardan ilki donanımla ilgili olup, dağıtık sistemi oluşturan bilgisayarların bağımsız olduğudur. İkincisi ise yazılımla ilgilidir ve dağıtık sistemin kullanıcıya tek bir bilgisayarmış gibi göründüğüdür [19].

Dağıtık sistemlerin temel özelliklerinden bir taneside ağın varlığının ve işlevinin kullanıcıya görünmemesidir. Çok işlemcili bir dağıtık sistem çalışmakta olmasına

rağmen, kullanıcıya görünen ise tek bir sanal işlemcidir. Dağıtık sistemlerin temel amacı kaynakların olduğunca etkin ve doğru paylaşımıdır.

1.1.1 Dağıtık Sistemin Özellikleri

Dağıtık bir sistemin genel özellikleri maddeler halinde aşağıda açıklandığı gibidir.

Dağıtık sistemlere verilen problemin tane boyutu (fine, medium, coarsegrain): Dağıtık sistemlerde tane boyutu (grain size) hangi seviyede paralellik yapacağının bir ölçüsüdür. Küçük taneli (fine-grain), yönerge (instruction) seviyesinde verilen işlerin paralel çalıştırılmasıdır. Paralel bilgisayarlarda küçük ölçekte paralellik görülmektedir. Büyük taneli (coarse-grain), program seviyesinde paralelliktir. Büyük taneli paralellikte aynı anda farklı programlar farklı işlemcilerde çalışmaktadır. Grid sistemlerinde büyük taneli paralellik daha uygun olmaktadır. Yerel ağlarda çalışan sistemlerde (küme bilgisayarlar) orta taneli (medium-grain) paralellik uygulanmaktadır [8].

İşleri dağıtma yöntemi (merkezi, dağıtık):

Dağıtık sistemlerde kullanıcılar tarafından sisteme verilen işlerin çalıştırılacakları işlemcilere dağıtma şeklini ifade etmektedir. Merkezi iş dağıtmada işler tek bir işlemciye (sunucuya) verilmektedir. Sunucu üzerindeki iş dağıtma mekanizması işleri çalıştıracakları işlemcilere göndermektedir. Merkezi yapılarda verilen işlerin hangi aşamada oldukları, tamamlanıp tamamlanmadığı gibi iş durumları da merkezde (sunucuda) takip edilmektedir. Dağıtık sisteme verilen işler önerdiğimiz çalışmada olduğu gibi birden fazla noktadan da dağıtık olarak işlemcilere dağıtılabilmektedir. İşler işlemcilere herhangi bir işlemci üzerinden verilebilmektedir. İşlerin çalışma anındaki aşamaları takip edilmemektedir [8].

Heterojen yapıların desteklenebilmesi:

Geliştirilen bazı sistemler sadece homojen yapıları (küme) desteklemektedir. Bazıları ise heterojen yapılara da destek vermektedirler. Heterojen yapılar sistemin büyüklüğünün değiştirilmesinde esnek çözümler sunmaktadırlar. Grid sistemlerin çoğu heterojen bilgisayarları desteklemektedir [8].

1.1.2 İdeal Dağıtık Sistemler

Dağıtık sistemlerde işlerin uç işlemcilere atanması dağıtık olmalıdır. Dağıtık sisteme kullanıcılar tarafından verilen işler tek bir noktadan (merkezden, sunucudan) değil de birden çok noktadan verilebilmelidir. Bütün kullanıcılar sisteme iş verebilmelidir. Kullanıcıların iş verebilmek için kayıtlı olmasına gerek olmamalıdır. Sistem içindeki uç bilgisayarlar isteğe bağlı olarak çalışabilmelidir. Uç bilgisayarlar istedikleri zaman dağıtık sisteme dâhil olup verilen işleri yapabilmeli, istedikleri zaman sistemden ayrılabilmelidir. Farklı iş çizelgeleme yöntemleri desteklenmelidir. Uç bilgisayarların dâhil olması veya ayrılması için kayıt mekanizmasına gerek olmamalıdır.

Dağıtık sistemlerin donanım, işletim sistemi, platform gibi gerçekleme kısıtlarının olmaması istenmektedir. Bunları destekleyebilmek için sistemin heterojen yapıları desteklemesi beklenmektedir. Dağıtık sistemler diğer sistemlerle bütünleşmiş olmalıdır. Dağıtık sistemler birden fazla ağda çalışabilmelidir. Dağıtık sistemlerde kullanıcı doğrulama ve güvenlik mekanizmaları desteklenmelidir. Dağıtık sistemler hataya duyarlı (fault tolerant) olmalıdır. Sisteme verilen işler çalışan diğer işlemciler tarafından başarı ile tamamlanmalıdır. Sisteme verilen işleri yapabilecek en az bir işlemci olduğu sürece tüm işler başarı ile tamamlanmalıdır.

1.2 Dağıtık Sistemlerde Yük Dengeleme

Dağıtık sistemlerin işlem kapasitesinin avantajlarından etkin bir şekilde yararlanabilmek için iyi bir kaynak dağıtım modeline ihtiyaç duyulmaktadır. Dağıtık işlem programlayıcısı, sistemin genel performansını en üst seviyede tutacak şekilde sistemin yükünü adil ve saydam bir şekilde işlemciler arasında dağıtır.

Dağıtık sistemlerde, sistem homojen bir yapıda olsa bile işlerin, yükü ağır işlemcilerden hafif yüklü işlemcilere transferi sistemin genel performansını

arttırmaktadır. Bu noktada sistem performansının nasıl belirleneceği sorusu akla gelebilir. Sistem performans ölçütlerinden biri işlemlere verilen ortalama cevap verme süresi, yani bir işin sisteme yüklenip, sonucunun geri dönmesine kadar geçen süredir. Yük dağıtımının amacı sıklıkla bu süreyi düşürmektir. Sistem performansını arttırmak bir yük dağıtımı modelinin ana hedefi olmakla birlikte, karşılanması gereken birkaç önemli gereksinim daha bulunmaktadır.

Ölçeklenebilirlik: Yük dağıtım modeli, daha büyük dağıtık sistemlerde de çalışabilmelidir. Bu minimum maliyetle hızlı dağıtım kararları alabilmeyi gerektirir.

Lokasyon bağımsızlığı: Transfer edilen işlem, transfer edildiği işlemciye göre farklı sonuçlar üretmemelidir. Yani transfer edilen işlemci de ürettiği sonuç ile orjinal işlemcide çalışması halinde üreteceği sonuçlar aynı olmalıdır.

Heterojenlik: Yük dağıtım modeli, farklı mimarilerde, değişik donanımlarda çalışabilmelidir.

1.2.1 Dinamik Yük Dengeleme

Dinamik yük dengeleme, mevcut sistemin durumuna tepki vererek dinamik iş yükünü dengelemek amacıyla çalışır. Dinamik yük dengeleme algoritmaları, küme durumu hakkında bilgi toplamakta ve bu bilgilere dayanarak proses tahsisine dair kararlar vermektedir. Dinamik algoritmalar heterojen yapıdaki bilgisayarlar üzerinde de çalışabilmektedir. Dinamik yük dengeleme adımları şu aşamalar ile gerçekleştirilmiştir:

- yük ölçümü,
- bilgi değişimi,
- proses transferi,
- seçim,
- yerleştirme.

1.2.1.1 Yük Ölçümü

Yük bilgisi hizmeti, yük dengelemenin en temel öğesidir. Her bir dinamik yük dengeleme algoritması yük bilgisi üzerine kuruludur. Düğümlerin yük dengelerinin tutarlı olabilmeleri için, yük bilgilerinin yük tahmini politikasına göre toplanmalıdır ve analiz edildikten sonra bu veriler tüm düğümler arasında değiştirilmelidir.

Yük ölçümü, her düğüm tarafından sahip olduğu iş yükü, kuyrukta bekleyen iş sayısı, kullanılan bellek miktarı ve ağ durumu gibi yük parametreleri göz önüne alınarak belirlenmektedir.

1.2.1.2 Bilgi Alış Veriş

Dinamik yük dengeleme algoritmalarının ikinci adımı olan bilgi politikası işlemi, tüm sistemin ya da yerel komşulukların yük durumu hakkında bilgi sağlamak için bilgi alış verişinin nasıl gerçekleştirileceğinin belirlenmesidir. Bu işlem dinamik yük dengeleme işlemleri haricinde kaynak takibi ve sistem izleme gibi amaçlar ile de kullanılmaktadır [21].

Bu aşamada dikkat edilmesi gereken husus sistem karmaşıklığını göz önüne alarak, olabilecek en uygun, ölçeklenebilir ve düşük ağ yüküne sahip bilgi politikasını belirlemektir.

Günümüzde kullanılmakta olan bilgi alış veriş politikaları, uygulandıkları veri iletim yöntemlerine göre periyodik, talep odaklı ve durum değişim odaklı olmak üzere üçe ayrılmaktadır.

Periyodik politikalar: Belirli zaman aralığında, tüm düğümler kendi durum bilgilerini yayın yaparlar. Periyodik yöntemin temelinde, yük verisinin iletileceği zaman aralığının belirlenmesi yer almaktadır. Bu süreç paralel sistemin büyüklüğüne ve yük durumuna göre belirlenmektedir. Zaman aralığının uzun tercih edilmesi durumunda,

yük dengeleme işleminin eski veriler ile gerçekleştirilmesi ve dolayısıyla yanlış kararlar alınarak sistem hızlanmasının olumsuz etkilenmesi söz konusu olacaktır. Öte yandan, kısa aralıkların tercih edilmesi durumunda ise, gereğinden çok yük durum bilgisi aktarımı yapılacak ve paralel sistemin üzerinde çalıştığı ağ, yük dengeleme algoritmasının üretmiş olduğu ve kullanılmayacak birçok veri taşıyarak, sistemin genel işleyişini olumsuz yönde etkileyecektir.

Talep odaklı politikalar: Bu politikalar altında, yük dengeleme algoritmasını çalıştıracak olan düğüm, işlemi başlatmadan önce tüm sisteme yük durum bilgisi alma talebinde bulunmaktadır. Talep mesajını alan her çalışma düğümü, yük durum bilgisi oluşturarak ilgili düğüme iletmektedir. Böylelikle en aza indirilmiş olan yük durum mesajları ile ağ, dinamik yük dengeleme algoritması verilerinin oluşturacağı olası aşırı yüklemelerden korunmuş olmaktadır. Ancak bu durumda her çalışma düğümünün yük bilgisi oluşturup, oluşturduğu veriyi geriye iletmesi belirli bir gecikme süresi alacağından yük dengeleme işleminin gerçekleştirilmesi gecikecektir. Ayrıca periyodik olarak gönderilen yük bilgisi talep mesajları, bir eşleme noktası olarak işlem görmektedir. Bu durum paralel sistemlerin çalışma hızını olumsuz etkilediği için istenmeyen bir durum oluşturmaktadır.

Durum değişim odaklı politikalar: çalışma düğümünün iç durumuna bağlı olarak oluşabilecek özel durumlar doğrultusunda, düğümlere bilgi vermektedir. Eş zamansız bir özellik sergileyen olay tabanlı yapıda mesajların, düğümlerde oluşabilecek hangi özel durumlarda üretileceği sorusuna yanıt verilmesi zorunluluğu bu yöntemin temel sorununu oluşturmaktadır.

1.2.1.3 Proses Transferi

Bilgi değişimlerinden sonra, merkezi bir düğüm veya tüm düğümler birbirleri hakkında yük bilgisi edinmiş olacaklardır. Bu bilgiler ışığında, düğümlerin yüklü veya yüksüz oldukları kararlarına varılacaktır. Var olan bir çok proses değişim

politikası eşik değeri odaklıdır. Farklı yük dengeleme algoritmaları, eşik değerinin belirlenmesi için farklı bilgileri kullanmaktadır.

Proses transfer politikaları önceden tanımlanmış veya dinamik olarak değişen tek eşiğe veya çift eşik değerlerine dayalı olabilir. Proses transfer politikası bir düğüm üzerindeki yükü belirler ve prosesi aktarmak için karar verir.

1.2.1.4 Yük Seçimi

Seçim politikası hangi prosesin transfer edileceğinin kararını verir. Bu proses başlatmayı bekleyen bir proses olabileceği gibi zaten çalışmakta olan bir proses'te olabilir. Çalışmakta olan bir proses'in aktarılması karmaşık olduğu için pek tercih edilmez.

1.2.1.5 Yerleşim

Yerleşim politikası, hedefteki düğümü seçmektedir. Seçilen düğümün az yüklü olması gerekmektedir. Bu işlem sonucu yük dengeleme işlemi sonlanmış olmaktadır.

1.2.2 Dinamik Yük Dengeleme Algoritmaları

Yük dengeleme algoritmaları, karar verme mekanizmasının sistem genelinde aldığı konuma göre sınıflandırılabilmektedir. Aşağıda, bu sınıflandırma üzerinde durularak, var olan yük dengeleme algoritmalarından bazıları açıklanmıştır [22].

1.2.2.1 Merkezi Yük Dengeleme

Var olan ya da yeni üretilen işlerin çalışma birimlerine tek bir merkez üzerinden aktarıldığı yöntemlerdir. Merkezi bir çalışma biriminin, işleri uç çalışma birimlerine dağıtması ve işini bitiren düğümlere mevcut işlenmemiş işlerin ataması ile gerçekleştirilebileceği gibi tüm çalışma düğümlerinin çalıştıracakları işleri merkezi bir iş havuzundan talep etmeleri ile de gerçekleştirebilir.

Bu yöntem ile iş birimlerinin, ortada işlenmeyi bekleyen bir iş olduğu sürece hiçbir zaman boşta beklememeleri sağlanmış olur. Bunun için, iş biriminin iş kuyruğunda bulunan mevcut iş miktarının, önceden belirlenen bir eşik değerinin altına düşmesi ile merkez düğümden ya da havuzdan yeni iş talep etmesi eylemi tetiklenmektedir. Bu yöntem beraberinde, merkezi iş düğümünün çalışmasının problem çözümünden bağımsız olmasını getirir. Bu da, bir iş düğümünün sürekli olarak problemden bağımsız çalışmasını ve toplam hızlanma ve etkinlik gibi paralellik ölçütlerinin olumsuz etkilenmesine neden olmaktadır.

Beraberinde getirdiği performans kısıdının haricinde, merkezi yöntemler düşük ölçeklenebilirliğe de sahiptir. İş düğümlerinin oluşturduğu ağ yapısı büyüdükçe, merkezi düğüm hem ağ bağlantısı yönünden hem de işlem kapasitesi yönünden darboğaz oluşturmakta ve merkezi yöntemlerin performans üzerindeki olumsuz etkisini daha da arttırmaktadır. Ancak, tüm performans engellerine rağmen kolay programlanabilmesi ve karmaşık olmayan yapısı ile birçok sistemde kolaylıkla uygulanmakta ve kullanılmaktadır [27].

1.2.2.2 Hiyerarşik Yük Dengeleme

Bu yapıda tüm işler ana işlem düğümü tarafından yönetici olarak işlem gören çalışma düğümleri arasında paylaştırılır. İşlerin pay edildiği yöneticiler ellerindeki işi yük durumlarına göre emirleri altında bulunan çalışma düğümlerine iletirler. İşlem

sonunda oluşturulan veri önce yöneticilerde toplanır, daha sonrada ana işlem düğümüne aktarılır. Oluşturulan bu yapı hiyerarşik bir yönetim oluşturmaktadır.

Hiyerarşik dinamik yük dengelemede amaç, yükün hiyerarşik yapının kolları arasında eşit olarak dağıtılmasının sağlanmasıdır. Bunun için hiyerarşik yapının yüksek numaralı seviyelerinde bulunan ve birbirine doğrudan ya da dolaylı olarak bağlanmış iş düğümlerinin birbirlerine iş iletmeleri ile yeterli olacaktır. İş iletimi, işin hiyerarşik yapıda en son seviyelerden üst seviyelere doğru taşınması ve buradan tekrar yükün çalıştırılacağı son seviyelere indirilmesi ile gerçekleştirilmektedir.

Hiyerarşik yük dengeleme işleminde diğer yöntemlerden farklı olarak, iş verisi iletimi yerine çalışma düğümü iletimi de gerçekleştirilmektedir. Bunun için yoğun çalışan yönetici düğüme, boş durumda olan yönetici düğümlerinden, çalışma düğümü transferi yapılır. Transfer işlemi, yönetici düğümde var olan alt düğüm tablosunun güncellenmesi ile gerçekleştirilir. Hesaplama ve veri birleştirme işlemi sırasında eşleme gerçekleştiren bu yöntem, büyük veri iletimi yerine küçük bir tablo güncellemesi gerektirdiği için çok daha hızlı bir yük dengeleme işlemi sağlamaktadır.

1.2.2.3 Dağıtık Yük Dengeleme

Dağıtık yük dengeleme yaklaşımlarında, karar stratejisinin belirlenmesi sorumluluğu düğümler arasında pay edilmektedir. Her düğüm elinde bulunan fazla yükün gideceği hedef düğümü belirlemekle ve yükün gönderilmesi işlemini gerçekleştirmekle yükümlüdür. Bu noktada ortaya konulan ayrım ise, iş biriminin elinde bulunan fazla yükün nihai hedefinin karmaşık algoritmalar kullanılarak tespit edilmesinin ne derece gerekli olduğu ile alakalıdır. Eldeki yükün gideceği son çalışma düğümünü belirledikten sonra bu düğüme iletimi gerçekleştirmek, yük dengesizliğine doğrudan bir çözüm getirmektir. Ancak bu yaklaşım, iş düğümünün çalışma süresinin ciddi bir oranda artmasına neden olabilmektedir.

Bir diğer yaklaşım da fazla yükün doğrudan komşuları arasında belirlenen bir mantık çerçevesinde iletilmesi işlemidir. Böylelikle hem topolojik olarak iş düğümüne en yakın ve en hızlı etkileşime girilecek düğüm ile irtibat kurulmuş olmakta, hem de hedef belirleme işlemi kısa sürede sonlandırılmaktadır. Ancak bu yaklaşım beraberinde, yük dengesizliklerinin birden çok ardıl adımda çözülebilmesini getirmektedir.

Bu çerçevede, dağıtık yük dengeleme yöntemleri kendi içlerinde doğrudan yük dengeleme ve ardıl yük dengeleme olmak üzere iki gruba ayrılmaktadır.

1.2.2.4 Doğrudan Yük Dengeleme

Doğrudan yük dengeleme yöntemlerinde, var olan fazla iş yükünün gönderileceği son iş düğümü doğrudan belirlenerek yük iletimi gerçekleştirilmektedir.

Dinamik Eşik Değerli Nesne Tabanlı Yük Dengeleme Algoritması: Bu doğrudan dinamik yük dengeleme algoritmasında veri parçalama ve metot kopyalama temel alınarak veri paralel ve nesne tabanlı bir paralel sistem ortaya konulmuştur. Yük dengeleme, bölüm nesneleri olarak adlandırılan, dengelenmesi gereken fazla işleri ve her çalışma biriminde kopyalanmış olarak bulunan metodları içeren nesnelerin, çalışma düğümleri arasında dinamik olarak belirlenen eşik değerine ve ortaya konan yeni bir hedef bulma algoritmasına göre iletilmesi ile gerçekleştirilmektedir. Yük dengeleme işleminin gerekliliğine ve hangi düğümler arasında yapılacağına dinamik olarak belirlenen eşik değerleri ile karar verilmektedir [24].

Öngörüye Dayalı Nesne Tabanlı Dinamik Yük Dengeleme Algoritması: Bu yöntemde bir önceki yöntem gibi, veri parçalama ve metot kopyalama işlemlerine dayanan bir nesne tabanlı paralel sistemde çalıştırılmak üzere tasarlanmıştır. Bir önceki yaklaşımdan farklı olarak hedef düğüm belirleme işleminde, her düğümün metodlarının çalışma sürelerinin gözlenmesi sonucu istatistiksel bir sonuç ortaya konup bu doğrultuda bir sonraki adımda en az işe sahip düğümün öngörü yolu ile

saptanması sağlanmaktadır. Bunun haricinde iletilecek verinin işlem hızı ile gönderim hızları karşılaştırılarak ortaya koyacağı ağ yüküne göre gönderim işleminin yararlılığı test edilmektedir [25].

Gönderici düğüm tarafından başlatılan dinamik yük dengeleme işleminde düğümlerinin güncel yük durumlarının tespiti için, işlenmeden beklemekte olan bölüm nesnelerinin sayısı ve her bir işin hesaplanması için öngörülen zaman kullanılırken, oluşan yük bilgilerinin iletimi periyodik bilgi iletişim politikası kullanılarak gerçekleştirilmektedir.

Algoritma, karar verme aşamasında çalışma düğümünde çalışan tüm metodların geçmiş bilgisini tutarak elde edilen istatistiksel yaklaşım ile bir sonraki metodun çalışma süresini tahmin etmeye çalışmaktadır. Bu işlem çalışma zamanı bilgisinin oluşturulması ve gelecek davranış öngörüsünde bulunulmasından oluşan iki adım ile gerçekleştirilir.

Alıcı Tarafından Başlatılan Dinamik Yük Dengeleme Algoritması: Yük dengeleme işleminin kimin tarafından başlatılacağı kararı, dağıtık yük dengeleme yaklaşımlarının başarımlarında etkileyici bir unsurdur. Yük dengelemenin fazla yüke sahip düğüm tarafından ya da boştaki düğüm tarafından başlatılması, yük dengesinin sağlanması için gerekli mesajlaşma sayısında önemli değişikliklere neden olabilmekte ve işlem hızını belirgin bir biçimde etkileyebilmektedir. Bu bölümde incelenen yaklaşım, dengeleme işleminin boştaki düğüm tarafından başlatılmasını sağlamakta, böylelikle iletilecek işin parçalanması ve yoğun düğümün belirlenmesi işlemlerini ön plana almaktadır [23][26].

Alıcı tarafından başlatılan yük dengeleme işlemi, işini bitiren her bir düğüm kendisi için en uygun vericiyi periyodik olarak yollanan yük durumu bildirim mesajları ile tespit etmesi ile yük transferi safhasına geçer. Yük verisi olarak düğümlerde çalıştırılan en son işin çalışma süresi göz önüne alınmaktadır. Buradaki amaç uzun süren işlemlerin bulunduğu iş düğümünde, diğer işlerin de uzun zaman alacak işler olduğu varsayımından yola çıkarak, olabilecek yük dengesizliklerinin önüne geçebilmektir.

Her düğüm, çalıştırdığı işlerin başlangıç ve bitiş sürelerini bir merkeze yollayarak, yine o merkezden yük yoğunluğuna göre azalarak sıralanmış olan düğümlerin yük durum tablosunu alırlar. Tek merkez üzerinden yük yoğunluğu takibi yapılması çalışma düğümü sayısının artışı ile genel işleyişi yavaşlatmakta ve bu yöntemin ölçeklenebilirliğini kısıtlamaktadır.

Bu yaklaşımda, hedef seçimi kadar görevin parçalara ayrılmasına da önem verilmektedir. İşin küçük parçalara ayrılması, iş düğümlerinin, her işin sonlanmasından başlayarak yeni işin işletilmesine kadar geçecek zamanda oluşacak kayıptan ötürü paralel sistemin etkiliğini azaltıp, işlemin hesaplanması için gerekli düğüm sayısını artırmaktadır. Büyük iş parçalarının kullanımı ise yük dengesizliğini artırarak paralel sistemin hızlanmasını olumsuz yönde etkilemektedir. Bu nedenle iş bölümü için dinamik bir mekanizma oluşturulmuştur. Bu mekanizma iş bölümü ve yük dengeleme ihtiyaçları arasından en uygun noktayı bulup görev parçalamayı ihtiyaca göre en uygun şekilde gerçekleştirmektedir [26].

1.2.2.5 Ardıl Yük Dengeleme

Ardıl dinamik yük dengeleme algoritmaları, iş yükü dağılımının, genel en iyi dağılımının ardışık tahminleri üzerinden iş görmektedir. Bu nedenle her adımda iş yükü aktarımını belirli boyutlarda ve sınırlar dâhilinde gerçekleştirilir. Fazla iş yükünün ulaştırılacağı son noktadan çok, o an için iş yükünün bulunması gereken eniyi iş düğümünü, komşu düğümler arasından bularak iş yükü iletiminin gerçekleştirilmesini sağlar.

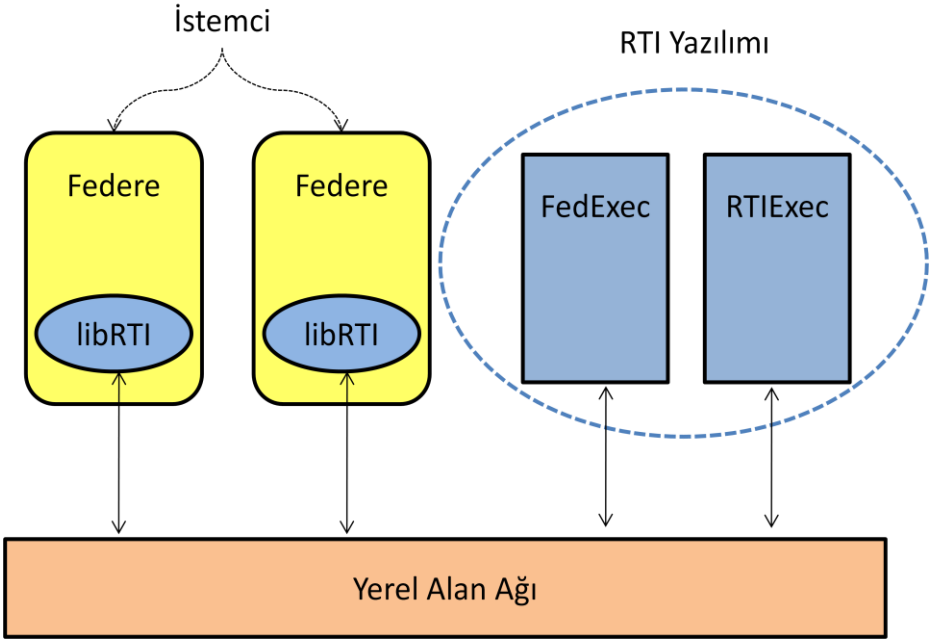
Bu yapıda, fiziksel olarak dağıtık yapıda ve belirli komşuluk mantığı ile birleştirilmiş çeşitli topolojilerdeki hesaplama sistemlerinde, komşular arası iletimin, doğrudan komşuluğu olmayan düğümlere göre hızlı yapılması dolayısı ile iletimde zamandan kazanç sağlamaktadır.

1.3 Yüksek Düzeyli Mimari

HLA dağıtık simülasyonların tasarlanması ve koordine edilmesinde, genel amaçlı bir yöntem sağlamak için geliştirilmiştir. 2000 yılında HLA bir IEEE standardı haline gelmiştir. Simülasyonların tasarımında tutarsızlığı önlemek ve birlikte çalışabilirliği ve yeniden kullanılabilirliği sağlamak üzere geliştirilmiştir.

Simülasyon sistemleri birbirinden farklı özelliklerde olan simülasyon bileşenlerinin biraraya gelmesiyle oluşmaktadırlar. Daha önce geliştirilmiş bileşenler yeni geliştirilen benzetim sistemleri ile uyumlu olması ihtiyacı doğmaktadır. HLA, sıfırdan büyük bir simülasyon sistemi kurulması yerine, var olan sistemlerin birleştirilerek yeni sistemler oluşturabilmesine olanak sağlamaktadır. HLA, aynı zamanda farklı işletim sistemlerine ve programlama dillerine sahip bileşenlerin kolayca birbirleriyle çalışabilecekleri bir ortam sağlamaktadır.

Benzetim sistemleri çoğu zaman farklı geliştirme grupları tarafından farklı teknolojiler kullanılarak geliştirilmiş ve değişik platformlarda çalışan diğer benzetim sistemleriyle birlikte çalışması gerekmiştir. Farklı teknolojilerin kullanımı ve farklı bilgi birikimlerinin ortak kullanımı ile benzetim sistemlerinin yetenekleri önemli ölçüde artmıştır. Örneğin C++ dili kullanılarak geliştirilen Unix platformu üzerinde çalışan bir uçak benzetim sistemi ile Java dili kullanılarak gerçekleştirilen, Windows platformunda çalışan bir uçaksavar benzetim sisteminin bir senaryo dâhilinde birlikte çalışabilmesi gibi.



Şekil 1.1: HLA Genel Mimarisi

Benzetim sistemleri geliştirilirken ortak bir belirtime uyulmazsa, önceki iki paragrafta bahsedilen yeniden kullanılabilirlik ve birlikte çalışabilirliği sağlamak için uygulama kodlarında büyük değişiklik yapmak gerekir. Çoğu zaman değişiklik yapma maliyeti, uygulamayı tekrar geliştirme maliyetine yakındır. HLA, bu sorunu çözmek için geliştirilmiş bir standarttır.

1.3.1 IEEE Std. 1516:Yapısı ve Kuralları

Bu standart HLA hakkında genel bilgi vermektedir. HLA konseptleri, HLA federasyonları ve federeleri üzerinde bir dizi kurallar uygular. Bu kurallar tüm HLA federeleri ve federasyonlar tarafından uyumlu olmalıdır. HLA kuralları, beşi federe, geriye kalan beşi de federasyon kuralı olmak üzere on kuraldan oluşur [1].

Federasyon kuralları;

- Federasyonun HLA nesne modeli şablonuna (Object Model Template - OMT) uygun bir federasyon nesne modeli (Federation Object Model - FOM) olması gerekir.
- FOM'daki nesnelerin tanımları RTI tarafından değil, federeler tarafından yapılmalıdır.
- Federasyon çalışması esnasında tüm FOM bilgi değişimi RTI üzerinden olmalıdır.
- Federasyon çalışması esnasında federelerle RTI arası iletişim HLA arayüz belirtimine uygun olarak yapılmalıdır.
- Federasyon çalışması esnasında bir olgu niteliğinin sahipliği aynı anda sadece bir federede olmalıdır. Sahiplik, çalışma zamanında federeler arasında el değiştirebilir.

Federe kuralları;

- Federenin HLA nesne modeli şablonuna göre düzenlenmiş bir benzetim nesne modeli (Simülation Object Model - SOM) olmalıdır.
- Federeler SOM'da tanımlanan kurallara uyarak olgu niteliklerinin değerlerini günleyip / yansıtabilirler, etkileşim gönderip / alabilir.
- Federeler SOM'da tanımlanan kurallara uyarak federasyon çalışması esnasında olgu niteliklerinin sahipliğini kabul edebilir / devredebilir.
- Federeler SOM'da tanımlanan kurallara uyarak sahip oldukları olgu niteliklerinin değerlerini günleyerek federasyonun akışını değiştirebilir.
- Federeler yerel zamanlarını federasyon üyesi diğer federelerle veri değişimini yönetebilecek şekilde kontrol edebilir.

1.3.2 Arayüz Belirtimi ve RTI

Arayüz belirtimi, federeler ve RTI arasındaki işlevsel arayüzleri tanımlar. IEEE Std. 1516.1 olarak standart halini almıştır. Arayüz belirtimi, federelerin RTI üzerinden

federasyonla ve birbirleriyle hangi servisleri kullanarak etkileşeceklerini tanımlar. RTI, arayüz belirtimine uyan, HLA uyumlu benzetim sistemleri için gerekli servis arayüzlerinin gerçekleştiriminden oluşan bir altyapıdır. RTI, taşınabilirliği ve birlikte çalışabilirliği arttıran bir yapıdır [2].

Federeler arası etkileşim için RTI kullanımını öngören HLA arayüz belirtiminin altı temel amacı vardır:

1. Benzetim sistemi ile iletişim altyapısını birbirinden ayırır.
2. RTI, DIS, ALSP gibi eski standartların eksik yönlerini giderir.
3. Federasyonların oluşturulup yok edilmesini sağlar.
4. Nesne / etkileşim tanımlayıp yönetmeye olanak sağlar
5. Federasyon zaman yönetimi ile uyumlu çalışır.
6. Federeleri mantıksal gruplara ayırıp iletişim etkinliğini artırır.

Arayüz belirtimi altı yönetim alanından oluşur;

1. Federasyon yönetimi (Federation Management)
2. Tanımlama yönetimi (Declaration Management)
3. Nesne yönetimi (Object Management)
4. Sahiplik yönetimi (Ownership Management)
5. Veri dağıtım yönetimi (Data Distribution Management)
6. Zaman yönetimi (Time Management)

2 İLGİLİ ÇALIŞMALAR VE TEZ KONUSU

2.1 Federe Göç Teknikleri

Göç bir çok alanda olabilmektedir, paralel ve dağıtık hesaplamalarda proses göçü ve mobil ajanlar örnek verilebilir. Bir mobil ajan, çalışma kodu ve veriden oluşmaktadır, bağımsız olarak göç etmesi beklenir, farklı ortamlara uyum sağlar ve kendi koşum durumunu sorunsuz bir şekilde tekrar yükleyebilir. Böylece, mobil ajanların birlikte çalışabilirliği sağlanır. Ayrıca mobil ajanlar, dağıtık sistemlerde göç etkilerini en aza indirerek şeffaflığını arttırmaktadır.

Bir proses göçü üç aşamada ele alınmaktadır; müzakere, transfer ve yerleşme. Müzakere fazı, bir prosesi almak için karar verme ve gelen proses için kaynak tahsis etme aşamalarından meydana gelir. Transfer fazında, prosesin sanal boşluğu ve kaynak düğümdeki iletişim bağlantıları hedefe kopyalanır. Transfer fazında olduğu gibi, proses koşumu yerleştirme fazında kurtarılır ve ilgili parçalar taşıma tamamlandığında bilgilendirilir.

Federe göçü yaklaşımları proses ve mobil ajan teknikleri alanlarında bağdaşır, fakat bilgilerin nasıl taşınacağı, federelerin nasıl durdurulacağı ve ne kadar simülasyonun düzenleneceği konularında farklılıklar göstermektedir.

Luthi ve GrBomann büyük ölçekli dağıtık simülasyonlarda HLA kullanarak ve kendi kaynak paylaşım sistemini oluşturarak bir çözüm geliştirmişlerdir. Simülasyon sistemlerinde, proses aktarımı sırasında simülasyon yöneticisi ve iletişim federeleri kullanılmıştır. Yönetici, yeni lokasyonda federelerin başlatılmasını ve kayıt edilmiş durumlarının transfer edilmesini yönetir. İletişim federeleri, federelerin kendi durumlarını kaydetmek ve tüm simülasyonu dondurmak için görevleri dağıtmaktadır [14].

Zajac ve birlikte çalıştığı kişiler Grid ortamları üzerinde koşan bir HLA simülasyonu sunmaktadır. Böyle bir sistemin kurulması amacıyla, Grid servisleri ve HLA simülasyon arasında bir arayüz gibi çalışan aktarım kütüphanesi geliştirilmiştir. Aktarım kütüphanesi, HLA API'lerine erişmekte, kayıt edilmesine yardım etmekte, durumları tekrar yüklemekte ve yazar ile federelerin kodları arasında bir arayüz görevi görmektedir. Federe durumları, HLA metodları ile kayıt edilmekte ve tekrar yüklenmekte, federe transferleri GridFTP kullanılarak gerçekleştirmektedir [15].

Cai ve birlikte çalıştığı kişiler bir Yük Yönetim Sistemi geliştirmiştir. Büyük ölçekli HLA bazlı dağıtık simülasyonlarda koşumu desteklemektedir. Bu sistemde ki en önemli özellik federe aktarımıdır. Bu sistemde aktarım sırasında tüm simülasyon durdurulmaktadır. Grid FTP servisleri ile federe aktarımı gerçekleştirilmektedir [16].

Sunulan tüm çözümler HLA standartları kullanılarak kaydetme ve tekrar yerleştirme işlemleri gerçekleştirmektedir. Ancak bu metodlar aktarım sırasında mesaj alış verişine izin vermeyerek simülasyon düzenini sağlamakta ve tüm simülasyonu durdurmaktadır. Ancak aktarım gecikmelerini en aza indirebilmek için simülasyonun durdurulmadan işlemlerin gerçekleştirilmesi gerekmektedir.

Tan ve Lim tarafından sunulan teknikte, HLA simülasyonlarında yük dengeleme sağlanmıştır. Bu mimaride bir federe kapsayıcısı (wrapper) ve yük dengeleme sisteminden oluşmaktadır. Yük dağıtım sistemi federeleri izlemekte ve federe aktarımını belirlemektedir. Federe kapsayıcısı SugarCube'ler kullanarak federe koşumlarını yönetmektedir. SugarCube'ler federeyi durdurmakta ve tekrar döndürmektedir. JavaGo2 aktarılan prosesin durumunu koruması için kullanılmaktadır. Ek olarak, aktarım sırasında gelen mesajlar bir kuyrukta tutulmakta ve özel simülasyon bölgesine, yayınlama ve abonelik yöntemiyle göndermektedir. Adı geçen teknikte üçüncü şahıs mekanizmalara ihtiyaç duyulmaktadır. Verileri göndermek için kullanılan araçlar aktarım zamanını arttırmaktadır [17].

Yuan ve diğerleri (birlikte çalıştığı kişiler) aktarım mekanizması, SimKernel üzerinde ve uygulama seviyesinde federe aktarımına odaklanarak, aktarım gecikmesini azaltmaktadır. Proses aktarımı sırasında, federeler durumlarını kayıt

eder ve kodları ile birlikte transfer edilir ve hedef düğümde gölge federeler başlatılır. Gösterilen çözümde simülasyon durdurulmamakta ve üçüncü şahıs mekanizmalar kullanılmamaktadır [18].

Tan ve birlikte çalıştığı kişiler aktarım gecikmesini azaltmak için, önerilen sistemde bir federeyi durdururken simülasyonu durdurmaz, federe durumları kayıt edilirken ve tekrar yerleştirilirken, veri transferi için eş düzeyde iletişim kullanılır. Çözümde yeni bir federe yaratılır ve federasyona kayıt edilir, federenin durumu yeniden yerleştirilir ve önceki nesne gibi aynı nesnelere yayın ve abonelikler gerçekleştirilir.

Yuan, Tam ve Lim tekniklerinde, her bir federenin yönetilebilmesi için gereksiz mesajlaşmalar gönderilmektedir. Bu yönetim yeni bir federenin sisteme katılmasını, eski federenin çıkartılmasını, aynı nesnelere abone ve yayın yapılması işlemlerini içermektedir. Tekrarlanan mesajlar iki federeyede gönderilir ve bu mesajların elenmesi için ek hesaplamalara ihtiyaç vardır. Ayrık HLA federeleri için iki aşamalı bir geçiş mekanizması sunan bu gereksiz yönetim ve bilgi işlem kısımlarından kaçınılmış olur. Bu ayrık federelerin göçü daha basit bir göç yönetimi gerektirir, çünkü tam olarak mesaj kayıpları ve düzenlenmesiyle ilgili değildir. Bu nedenle, göç sürecinde oluşan gecikmenin en aza indirilmesi ve bu tür sistemlerin başlıca hedefidir.

Bir başka referans kaynağı olan, Anders Persson'ın tezinde, HLA federasyonlarında yük dengesizliğini sezmek ve federe aktarımı yapmak konuları ele alınmıştır ve bu sistemde genel bir mimari ortaya konmuştur. Yapılan çalışmada HLA mimarisi üzerinde proses aktarımı konusu üzerinde durulmuştur. Bir diğer referans kaynağında iyileştirilmiş federe aktarım protokolü üzerinde durulmuştur. Sistemde geliştirilmiş federe aktarım tekniği, simülasyon ajanı bazlıdır (simulation agent), simülasyon ajanı federe aktarımı yönetimini kolaylaştırmak ve gecikmeleri azaltmak amacıyla simülasyonlar içerisinde kullanılmıştır. Robson Eduardo De Grande tarafından yapılan çalışmada, oluşturulan aktarım yükünü önlemek amacıyla, geliştirilmiş sistemde freeze-free federe aktarım tekniği kullanılmıştır. Durdurulmadan (Freeze-free) federe aktarımı ile simülasyonun aktarım sırasında durdurulmaması tekniğidir [10][12].

2.2 Tez Konusu Kapsamında Yapılması Planlanan Çalışmalar

Yapılacak olan çalışmanın konusu, HLA RTI bazlı simülasyonlar üzerinde dinamik bir yük dengeleme sisteminin geliştirilmesidir. Çalışmanın temel hedefi referans verilen kaynaklardaki yöntemler temel alınarak, dağıtık simülasyon ortamında daha güvenilir (mesaj kayıpsız) etkin bir yük dengeleme sisteminin geliştirilmesidir.

HLA bazlı simülasyonlarda performans kaybı koşum sırasında artmaktadır. Dağıtık sistemlerde yüksek olasılıkla birtakım bilgisayarlar aşırı yüklenmişken diğerleri boş veya az yüklenmiş olabilir. Bu da performans kaybına neden olur. Bunun önüne geçebilmek için yük dengeleme kullanılması gerekmektedir.

Bu çalışmada dinamik yük dengeleme stratejisi benimsenmiştir. Dinamik yük dengeleme stratejisinde, sistemde yer alan her bir düğüm üzerinde periyodik olarak bir yük dengeleme algoritmasının çalışması ve sistem kaynaklarının o anda ki kullanım değerlerine göre çalışmakta olan yükün, daha az yüke sahip farklı bir düğüm üzerine yeniden yerleştirilmesi çalışmalarını kapsamaktadır.

Yapılan çalışma dört temel aşamada ele alınmıştır. İlk olarak, Portico Projesinin sağlamış olduğu RTI servisleri ile uyumlu çalışan bir HLA bazlı simülasyon uygulaması geliştirilmiştir. Çalışmanın ikinci kısmında, yürütülmekte olan bir federenin bir düğümden diğer uzak bir düğüme aktarımı gerçekleştirilmiştir. Geliştirilen aktarım protokolünde hedeflenen bazı önemli noktalar vardır, bunlar; çalışan federasyonun durdurulmadan aktarımın gerçekleştirilmesi, aktarım sırasında oluşacak herhangi bir veri kaybının önlenmesi ve aktarımın çalışan diğer federelerin çalışmalarını engellemiyor olması. Bu aktarım Bölüm 3'te detaylı olarak anlatılmıştır. Çalışmanın üçüncü safhasında, dağıtık bir sistem üzerinde yük dengeleme konusu ele alınmıştır. Bu safhada verimli bir çalışma ortamının sağlanması için gerekli yük dağılımı hesaplanmıştır ve üç temel sorun üzerinde geliştirme yapılmıştır, bunlar; ne zaman aktarımın yapılacağı, hangi düğümden hangi uzak düğüme aktarım yapılacağı ve hangi federenin aktarılacağıdır.

Çalışmanın son safhası ise bu üç sistemin birbirleriyle entegre edilmesi ve gerekli test ortamlarının yaratılarak, elde edilen sonuçların değerlendirilmesi olarak tanımlanmıştır.

Çalışma kapsamında bazı kısıtlar öngörülmüştür, bunlar;

- Ağda bulunan düğümler (bilgisayarlar) heterojen (farklı donanım özellikleri) yapıda olabileceklerdir.
- Ağda bulunan bilgisayarların windows işletim sistemine sahip oldukları varsayılmıştır.
- Ağda bulunan her bir bilgisayar en az .NET framework 2.0 çalıştırabilir olduğu varsayılmıştır.
- Ağda bulunan bilgisayarlar üzerinde farklı uygulamalar yer alabilecektir, bu da yükü etkileyen faktörlerin yalnızca kendi üzerinde çalışan federasyon ile ilgili olmadığını göstermektedir.
- Federeler arası mesajlaşma eşzamansız şekilde gerçekleştirilecektir.

3 HLA BAZLI SİMÜLASYONLAR ÜZERİNDE FEDERE AKTARIMI

HLA simülasyonlarında yük dengeleme amacıyla federe aktarımı temel bir yaklaşımdır. Aktarım protokolü ile bilgisayar kaynakları arasında yük paylaşımı yapılmaktadır. Simülasyon yazılımlarında yük dengeleme yapılması sırasında, federe aktarımı aşamalarının iyileştirilmesi, sistemin genel performansı açısından önemlidir, çünkü her aktarım sırasında simülasyonda belirli bir zamanda gecikmeler meydana gelmektedir. Yapılan çalışmada oluşturulan aktarım gecikmelerini önlemek amacıyla federasyonu durdurmadan aktarım yapılması tekniği kullanılmıştır. Federe aktarımındaki temel yaklaşımımız, durum aktarımlı federe taşınması (DAFT) olarak adlandırılabilinir. Sistemde geliştirilmiş federe aktarım protokolü detayları aşağıda verilmiştir.

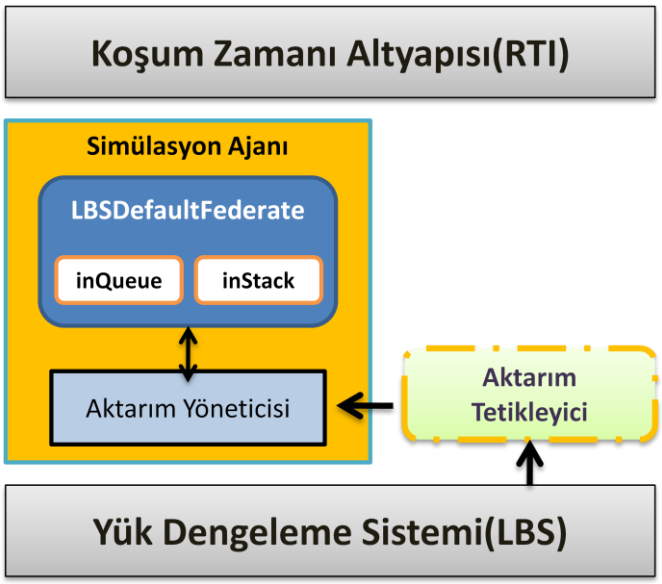
3.1 Durum Aktarımı ile Federe Taşınması (DAFT)

Sistemde geliştirilmiş federe aktarım tekniği, simülasyon ajanı bazlıdır (simulation agent), simülasyon ajanı federe aktarımı yönetimini kolaylaştırmak ve gecikmeleri azaltmak amacıyla federe durumlarının kayıt edilmesi ve durum aktarımı yapılması yöntemiyle her bir federe ile birlikte kullanılmıştır. Federe aktarım sürecinde, veri aktarımı için üçüncü parti (third party) yazılımları kullanılmamıştır ve daha önce yapılan çalışmaların aksine federe aktarımı için gölge federeler (shadow federate) kullanılmamıştır, bu sayede aktarım sürecinde sisteme getirilen ek yükten kaçınılmıştır.

DAFT yönteminde, aktarım iki temel aşamadan meydana gelmektedir. İlk aşamada çalışan federenin durumu kayıt edilir. Bir federenin durumu o federenin o anda sahip olduğu özniteliklerini ve özelliklerini ifade etmektedir. İkinci aşamada ise aktarım

için elde edilen durum bilgisi yeni bilgisayar düğümüne taşınır ve yeni konumda federenin eski durumuna getirilmesi ile simülasyona dahil edilir.

Sistemde aktarılan federeler, aktarım yönetimi tarafından yeni bilgisayar düğümünde simülasyon ajanı aracılığıyla yaratılmaktadır. Bu durum aktarım yapılan federenin bir bağ oluşturmadan tekrar aktarılabileceği anlamına gelmektedir. Şekil 3.1'de tasvir edildiği üzere, simülasyon ajanları RTI ile federeler arasında orta katmanda yer almaktadır. Federasyonda yer alan her bir federe için bir simülasyon ajanı yer almaktadır. Bu sayede federeler aktarım için herhangi ek bir yazılıma ihtiyaç duymamaktadır.



Şekil 3.1: Simülasyon Ajanı Mimarisi

3.1.1 Simülasyon Ajanı Mimarisi

Şekil 3.1'de görüldüğü gibi simülasyon ajanları yük dengeleme sistemi ve koşum zamanı altyapısı katmanı arasında yerleştirilmiştir. Simülasyon ajanları federelerin ne zaman taşınması gerektiği ile ilgilenmezler, ajanların görevi yalnızca yük dengeleme

sistemi tarafından istenilen yere federenin transferini gerçekleştirmektir. Aktarım yapabilmek için, simülasyon ajanı uzak düğümdeki simülasyon ajanını çalıştırmaktadır, ardından aktarımı gerçekleştirebilmek için bu iki simülasyon ajanı birbirleriyle haberleşmektedirler.

Simülasyon ajanları federelerin taşınması sırasında federelerin durumlarının kayıt edilmesi ve tekrar yüklenmesi görevlerini gerçekleştirirler, bu durum aşağıda detaylı bir şekilde verilmiştir. HLA üzerinde federeler arası iletişim mesajlar (interaction) aracılığıyla gerçekleştirilmektedir. Bu mesajlar bir federenin aktarılması sırasında RTI'a iletilmesi yerine bir kuyrukta (inQueue) tutulmaktadır. Tutulan bu mesajlar daha sonra aktarımı biten federeye taşınarak federenin yeni konumunda mesajların kaybolmadan federenin simülasyona devam etmesini sağlamaktadır.

Aynı zamanda iletişim için oluşturulan her bir mesaj simülasyon ajanı içerisinde sınırlı bir yığın (inStack) içerisinde tutulmaktadır. Simülasyon varlıkları birbirlerine mesaj gönderirlerken her bir mesaja id'ler vermekte ve mesajların bir kopyasını yığına atmaktadır. Bu yığın ileride oluşabilecek herhangi bir mesaj kaybını önlemede kullanılmaktadır.

Simülasyon ajanı içerisinde aktarım yöneticileri ayrı bir iş parçacığı (thread) üzerinde dinleme modunda beklemektedir. Aktarım yöneticisi yük dengeleme sisteminden gelen ilgili federenin taşınması isteklerini karşılamaktadır. Bu istekleri karşılamak için her bir aktarım yöneticisi birer adlandırılmış iletim yolu (named pipe) açmaktadır. Bu sayede direkt araya farklı bir katman eklemeden federenin kendisine aktarım isteği gönderilmektedir.

3.1.2 Durum Metodlarının Kayıt Edilmesi ve Tekrar Yüklenmesi

Tasarımda şeffaflık olmasına rağmen, HLA ve federeler üzerinde bazı değişiklikler yapılarak aktarım sırasındaki gecikmeler azaltılabilmektedir. HLA standartlarında arayüzler aracılığıyla sunulan federe yönetim servislerinden, federasyon kaydetme

(federation save) ve fedrerasyon tekrar yükleme (federation restore) çağrıları kullanılmamaktadır çünkü bu servisler tüm simülasyon durumunu kaydetme ve tekrar yüklemek için kullanılmaktadır. Bu servislerin kullanılması durumunda RTI tüm sistem bazında senkronizasyon yapacağı için performans kötü şekilde etkilenecektir. Daha önce incelenmiş olan sistemlerde olduğu gibi, geliştirilen sistemde de, federe kaydetme (federate save) ve federe tekrar yükleme (federate restore) metodları kullanılarak federe bazında kayıt ve tekrar yükleme gerçekleştirilmiştir. Bu sayede simülasyonun tümden durdurulmasının önüne geçilmiştir.

Federe aktarımı senaryosunda, aktarım sırasında ve "federateSave" metodunun çağrılmasından sonra, aktarımı yapılan federeler, gelen mesajların hesaplanması ve yeni mesajların (interaction) başka federelere gönderilmesi durdurulmaktadır, bu durumda sistem de gecikmeler meydana gelebilmektedir. Bu noktada dikkat edilmesi gereken bu gecikmenin aktarım sırasında en aza indirilmesidir.

3.1.3 Mesajların Kayıt Edilmesi ve Tekrar Yüklenmesi

Daha önce de belirtildiği gibi, tasarımda federe kaydetme ve yeniden yerleştirme işlemleri, simülasyon koşumu durdurulmadan gerçekleştirilmektedir. Bu teknik aktarım gecikmesini azaltsa da bazı komplike ve ek mekanizmaların tasarlanmasını gerektirmektedir. Bu yüzden, mesajlar bir veri yapısı içerisinde kayıt edilmeli, yeni konumda federelere gönderilmeli, yeniden yüklenmeli ve bütün bunlar tutarlı bir düzen içerisinde yapılmalıdır.

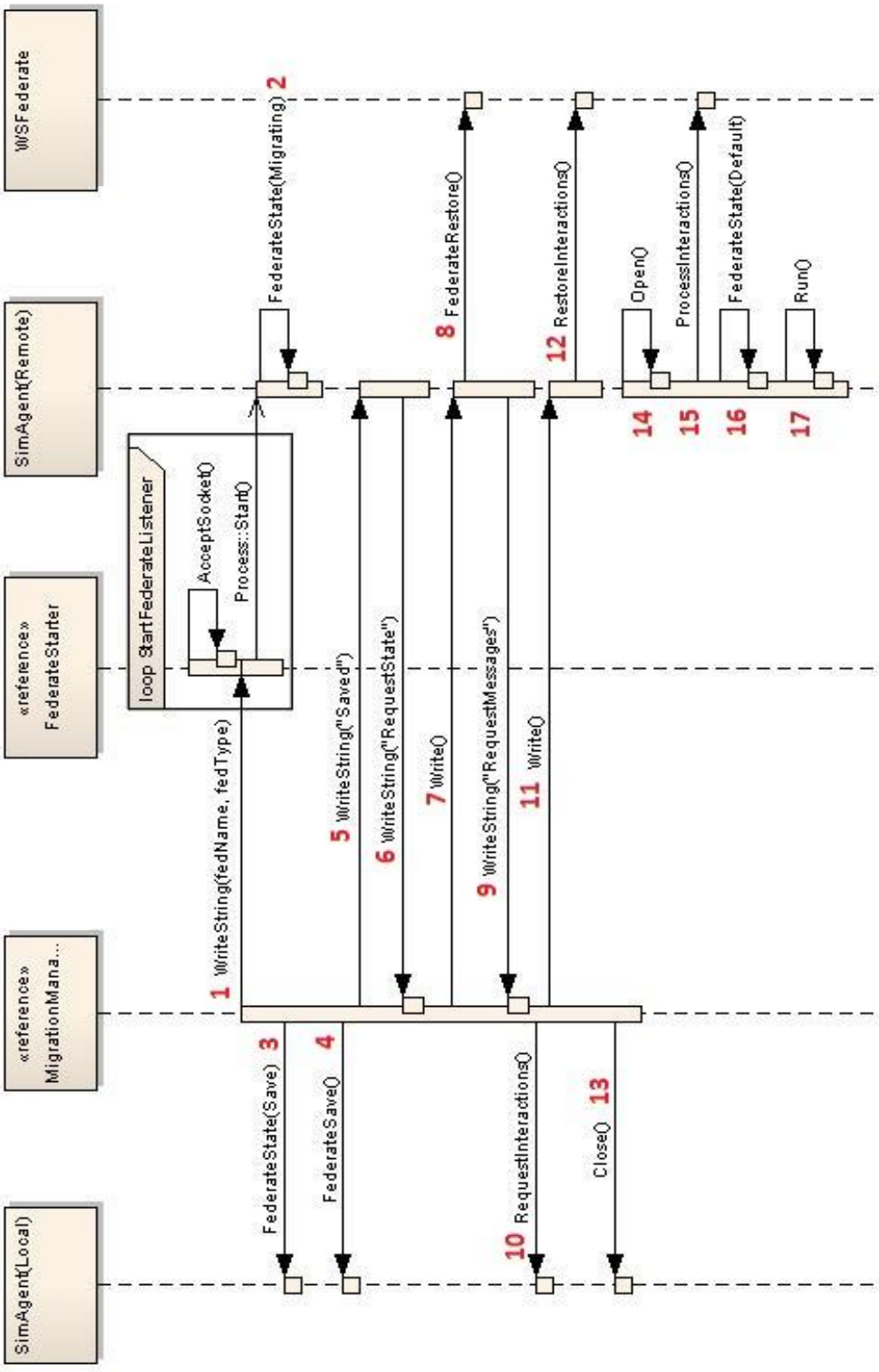
Aktarım sırasında tüm simülasyon durdurulmamaktadır, diğer tüm federeler mesajlaşmaya devam etmektedirler, yalnızca taşınan federe mesajlarını gönderilmek üzere kayıt etmiş ve taşındıktan sonra yeni konumda federeye düzgün bir şekilde yüklenmiş olmalıdır.

Tasarlanan yaklaşımda, tüm alınan mesajlar daha sonra işleme alınabilmeleri için bir kuyruğa kayıt edilmektedir. Kuyruk durum metodlarının aktarımı yaklaşımındaki gibi karşı federeye aktarılmaktadır.

Aktarım işleminde bir sonraki adım, alınan mesajların yeni lokasyondaki federeye gönderilmesi ve yeniden yerleştirilmesi işlemidir. Bu konuda yapılmış daha önceki çalışmalardan farklı olarak bu yaklaşımda, tüm mesajların kayıt edilmesinden ve hiç bir mesaj alınmayacağından emin olunduktan sonra, uzak lokasyondaki simülasyon ajanına tüm mesajlar paketlenerek gönderilirler. Yeni lokasyondaki ajan diğer elemanlara kendi taşınmış olan federenin aldığı son mesaj id'lerini mesaj olarak gönderir, mesajı alan diğer federeler kendi gönderim kuyruklarından mesajlarının ulaşıp ulaşmadığını kontrol ederler. Ulaşmayan mesaj var ise aktarım sırasında herhangi bir nedenden dolayı paket kaybolmuş ise bu mesaj tekrar gönderilir. Bu prosedür tamamlandıktan sonra yeni düğümdeki ajan mesajları kendi kuyruğuna ekler ve işletmeye başlar.

3.1.4 Federe Aktarım Protokolü

Şekil 3.2'de görülen diyagramda, federe aktarım prosedürleri, "Aktarım Tetikleyicisinin (Migration Trigger)", yük dengeleme sisteminden aktarım çağrısı alması ile başlar. "Aktarım Tetikleyicisi" çağrıldıktan sonra sırasıyla aşağıda tanımlanmış olan aktarım prosedürleri gerçekleşir.



Şekil 3.2: Federe Aktarım Protokolü

Aktarımın ilk adımında, "Aktarım Tetikleyici" ilgili federe ismiyle açılmış olan adlandırılmış iletim yolunu kullanarak, ilgili federenin "Aktarım Yöneticisini (Migration Manager)" başlatır, böylelikle federe aktarımı başlatılır. "Aktarım Yöneticisi", yük dengeleme sistemi tarafından belirlenmiş olan yeni lokasyonda, bir adet simülasyon ajanı başlatmak üzere "Federe Başlatıcıyı (Federate Starter)" çağırır. Yeni lokasyonda dinleme konumunda bekleyen "Federe Başlatıcı", gelen federe ismi ve federe tipinde yeni bir simülasyon ajanı yaratır. Bu uzak simülasyon ajanı kendini "Aktarım Bayrağını (Migration Flag)" ayarlayarak başlatır ve simülasyona dahil edilmez. Yeni lokasyonda başlayan simülasyon ajanı böylelikle aktarımın gerçekleşeceği süre boyunca, kayıt edilmiş durumda ve mesaj bekleyen modda bekletilir.

Aktarımın ikinci aşamasında, eski lokasyonda aktarılacak simülasyon ajanı kendi durumunu kaydetme konumuna (save flag) ayarlayarak, federe aktarım işleminin başladığı konusunda bilgilendirilir. Böylelikle eski lokasyondan federenin mesajları göndermesini keserek mesajların kuyrukta (inQueue) tutulmasını sağlar.

Üçüncü aşamada, "Aktarım Yöneticisi" tarafından ilgili federenin kayıt etme metodu çalıştırılarak federelerin koşumları durdurulur ve federelerin durumları kayıt edilerek ikili (binary) dosyaya yazılır. Böylece federe, kendi simülasyon döngüsündeki tüm işlemleri tamamladıktan sonra, kendi durumuna karşılık gelen tüm simülasyon değişkenlerini kayıt eder ve onlarla yöneticiye cevap verir ve kendi koşumunu durdurur.

Dördüncü adımda, eski lokasyondaki "Aktarım Yöneticisi", yeni lokasyondaki "Aktarım Yöneticisine", kendi durumunun kayıt edildiği mesajını gönderir. Yeni lokasyonda kayıt edildi mesajını alan "Aktarım Yöneticisi", kendi durumunu yeni lokasyonda tekrar yerleştirmek üzere, kayıt edilmiş olan federelerin durumlarını karşı taraftan ister. Eski lokasyondaki "Aktarım Yöneticisi" durumları gönderir. Yeni lokasyonda durumlar alındıktan sonra, "Simülasyon Ajanı", federenin "federe tekrar yükleme" metodunu çağırarak, federe durumlarını tekrar yerleştirir.

Beşinci adımda, yeni konumdaki "Aktarım Yöneticisi", aktarım sırasında gelen mesajları ister. Eski lokasyonda mesaj isteğini alan "Aktarım Yöneticisi", kuyrukta

biriken mesajları, yeni lokasyona gönderir ve koşumunu sonlandırır ve bu durum RTI'a bildirilir. Böylelikle federenin eski lokasyondaki varlığı tamamen kaldırılmış olur. Yeni lokasyonda alınan mesajlar simülasyon ajanında işletilmek üzere yerleştirilir. Aynı zamanda yeni lokasyonda federe başlatılır ve RTI'a bildirilir. Artık federasyonda bulunan tüm federeler, taşınmış olan federenin yeni lokasyondaki adresini bilir. Bu sırada yeni konumunda alınan mesajlar var ise işlenmeyerek kuyrukta tutulmaktadır. Yeni konumda mesajları alan federe, tüm federelere kontrol mesajı göndererek kendisine gelen son mesaj tekil numarasını gönderir. Taşınan federe eski ve yeni konumdaki mesajları birleştirerek işleme alır ve aktarım sırasında alamadığı mesajları almış ve gönderemediği mesajları göndermiş olur. Simülasyon ajanı durumunu varsayılana (default flag) çekerek federenin simülasyon döngüsüne girmesini sağlar.

Bütün bu adımlar bitirildikten sonra, diğer federeler kontrol mesajı ile aldıkları son numarayı mesaj yığınında kontrol eder ve gerekir ise taşınmış federe tarafından alınamayan mesajları tekrar gönderir. Böylece aktarımda meydana gelmiş veya gelebilecek kayıpların önüne geçilir.

4 YÜK DENGELEME SİSTEMİ

Yük dengeleme sistemi HLA simülasyonu üzerinde koşan federelerin ağdaki bilgisayarlar arasında orantılı bir şekilde dağıtılması amacıyla kurulmuş olan sistemdir. "Yük Yöneticisi (Load Manager)" ve "Yük İzleyicisi (Load Monitor)" olmak üzere iki bölümden oluşur.

Yük İzleyicisi (Load Monitor): Federelerin koştuğu bilgisayar düğümlerinin, CPU, bellek ve ağ yoğunluğu gibi sistem kaynaklarını izlemek ve bu bilgileri loglama yapmak ile görevli bileşendir. Yük izleyicisi belirli zaman aralıklarında loglama işlemini yürütmektedir.

Yük Yöneticisi (Load Manager): Karar verme mekanizmasını içeren bileşendir. Federe aktarımı yapılması gerekliliğini, hangi federenin aktarılacağı ve hangi düğüme taşınacağı kararlarını vermektedir. Yük dengelemesinde bir bilgisayarın çalışma performansını etkileyebilecek olan CPU kullanımı, CPU kuyrukta bekleyen iş sayısı, bellek kullanımı ve ağ iletişim yoğunluğu baz alınarak bir yük yoğunluğu hesabı yapılmıştır.

Algoritma 1'de görüldüğü gibi, yük dengeleme bir döngü içerisinde periyodik deltaT (ΔT) zaman adımında yürütülmektedir. Küçük adımlardaki çevrimlerde veri toplama ve karar adımları sebebiyle sistemin performansını kötü yönde etkileyebilmektedir. Algoritmada ΔT adımı asgari düzeyde yük üretiminin sağlanması ve yük değişimlerinin farkında olunabilmesi açısından 5 saniye olarak seçilmiştir.

Algoritmada her bir çevrimin ilk adımında veri toplama işlemi gerçekleştirilmektedir. Veri toplama işlemi sisteme dâhil olan tüm aktif düğümlerin yük izleyicisi tarafından toplanan verilere erişim ile gerçekleştirilir. Veri toplama işlemleri gerçekleştirildikten sonra, ağda bulunan kaynakların (her bir düğüme ait CPU, CPU kuyruk bilgisi, bellek ve ağ yoğunluğu) ayrı ayrı ağırlıkları belirlenir. Düğüm parametrelerinin ağırlıklarının hesaplanmasında, sabit olarak verilen yüksek ve düşük eşik değeri katsayıları kullanılır. Elde edilen yüksek ve düşük eşik değerleri doğrultusunda ilgili düğümün ilgili parametresi o düğümün yük durumunu ortaya

çıkartır. Elde edilen düğümün yük durumları hesaplama yapılan düğüm ile karşılaştırıldığında, o an hesaplanan yük durumlarına göre bir yük dengeleme işleminin başlatılıp başlatılmayacağı yani sistem içerisindeki kaynaklarda bir dengesizlik olup olmadığı kararı verilir.

Yük yönetiminde kullanılan algoritma aşağıda verilmiştir;

Algoritma 1 : Dağıtık Dinamik Yük Dengeleme Algoritması

```
1   ht = 1.2     /* Yüksek eşik değeri */
2   lt = 0.7     /* Düşük eşik değeri */
3   whileTRUEdo
4      activePCList= GetActivePCList()
5      remotePCUtilizations = GetRemoteUtilizations()
6      foreachpcUtilizationon remotePCUtilizations
7               list_cpu_u.Add(pcUtilization.cpuu)
8               list_cpu_q.Add(pcUtilization.cpuq)
9               list_mem_u.Add(pcUtilization.memu)
10              list_net_u.Add(pcUtilization.netu)
11     endfor
12
13     cpuAvg = ∑ list_cpu_u/nodeCount
14     cpuHT = cpuAvg * ht
15     cpuLT = cpuAvg * lt
16
17     cpuqAvg = ∑ list_cpu_q/nodeCount
18     cpuqHT = cpuqAvg * ht
19     cpuqLT = cpuqAvg * lt
20
21     memAvg = ∑ list_mem_u/nodeCount
22     memHT = memAvg * ht
23     memLT = memAvg * lt
24
25     netAvg = ∑ list_net_u/nodeCount
26     netHT = netAvg * ht
27     netLT = netAvg * lt
28
29     foreachnodeon activePCList
30              cpuState = GetLoadState(cpu_u)
31              cpuqState = GetLoadState(cpu_q)
32              memState = GetLoadState(mem_u)
33              netState = GetLoadState(net_u)
34
35              if((cpuState == IDLE) &&(mem_u<70))
36                      pcState = IDLE
37                      lowLoadedNodeList.Add(node)
38              elseif(((mem_u> 85)&&(cpuqState == HIGH )||(mem_u> 90) ||
39                                      (cpuState == HIGH  &&memState == HIGH))
40                      pcState = HIGH
41              elseif((cpuState == LOW)&&(netState == LOW )||(mem_u< 70) ||
42                                      (cpuqState == LOW  ))
43                      pcState = LOW
```

```
44                      lowLoadedNodeList.Add(node)
45              else
46                      pcState = NORMAL
47      endfor
48
49      if(pcState == HIGH)
50              ManageLOADS(lowLoadedNodeList)
51      wait(ΔT)
52  end
```

4.1 Yük Tahmini ve Bilgi Değişimi Politikası

İdeal olarak, yük bilgileri mevcut CPU kullanımı, bellek kullanımı ve bir düğümün ağ trafiğini yansıtmalıdır. Genellikle, bir düğümün yükü, verilen zamandaki CPU kuyruk uzunluğu olarak tanımlanmıştır. CPU kuyruk uzunluğu, çalışmakta olan veya çalıştırılmak için bekleyen proses'lerin sayısını ifade eder. Sistemde kullanılan parametreler CPU kullanımı, CPU kuyruk uzunluğu, bellek kullanım miktarı vede ağ trafiğidir. Örneğin, CPU kullanımı bir saniyede düşük iken diğer bir saniyede yüksek olabilmektedir. Bu nedenle Δt zaman aralığında ilgili parametre için ortalama değerleri alınarak bulunur. Verilen algoritmada 1 saniye aralıkla örnekleme alınmaktadır.

$$L_i(p) = \frac{p_{i1} + p_{i2} + \cdots + p_{ik}}{k}$$

- L_i = Δt saniyedeki ortalama yük parametresini ifade eder.
- p = yük bilgi parametresini ifade eder.
- $p_{i1}...p_{ik}$ = bir saniye aralıklardaki verilen değer parametrelerini ifade eder.
- k = örnekleme sayısını ifade eder.
- i = düğüm numarasını ifade eder (1, 2, 3 ... , n).

4.1.1 Yük Sınıflandırılması

İlk adımda her bir düğüm için yük sınıflandırılması yapılır. Verilen algoritma için dört adet yük seviyesi vardır: Beklemede (Idle), Düşük (Low), Normal ve Yüksek (High).

Bu adımın ilk kısmında, CPU kuyruk uzunluğu, CPU kullanımı, bellek kullanımı ve ağ trafiği için ikişer adet eşik değeri hesaplanır. İki eşik değeri kullanmanın sebebi daha istikrarlı sonuçlar elde edilebilmesidir. Bu dört parametre için eşik değeri hesaplanması aynı yöntem ile gerçekleştirilir:

1. Tüm düğümler üzerinde, her bir parametre için ortalama yük hesaplanır.

$$L_{Ort}(p) = \frac{l_1 + l_2 + \cdots + l_n}{n}$$

   - L_{Ort} = Verilen parametreler için tüm düğümler üzerindeki ortalama yük.
   - p = Yük parametresi, CPU kuyruk uzunluğu, CPU kullanımı, bellek kullanımı veya ağ kullanımı.
   - $l_1 \ldots l_n$ = yük tahmini politikası tarafından her bir düğüm için türetilen o anki yük parametesi.
   - n = düğüm sayısı.

2. Eşik değerleri için yüksek ve düşük çarpım katsayıları hesaplanması için standart sapma hesaplanır.

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(L_i - L_{ort})^2}$$

   - s = elde edilmiş yük parametreleri için standart sapma değeri.
   - L_i = yük tahmini politikası tarafından her bir düğüm için türetilen o anki yük parametesi. i : (1, 2, 3 ... , n)

- L_{Ort} = Verilen parametreler için tüm düğümler üzerindeki ortalama yük.
- N = düğüm sayısı.

3. Eşik değerleri hesaplanır.

   CPU kuyruk uzunluğu, CPU kullanımı, bellek kullanımı ve ağ trafiği, üst ve alt eşik değerleri; her bir parametre için ortalama yük ile sabit değerin çarpılmasından elde edilir.

$$t_H = (1 + s) * L_{avg}$$

$$t_L = (1 - s) * L_{avg}$$

- t_H = yüksek eşik değeri,
- t_L = düşük eşik değeri.

Verilen algoritmada yüksek ve düşük eşik değerleri ile belirli bir yük değeri parametresi için yüksek veya düşük yüklü olup olmadığı belirlenir. Bu iki eşik değeri arasında kalan değerler normal olarak kabul edilir. Bu sınıflandırmaların her biri kendi parametreleri için geçerlidir, düğümler için değildir. Yani bu hesaplama ve parametrelerin sınıflandırılması, CPU kuyruk uzunluğu, CPU kullanımı, bellek kullanımı ve ağ trafiği içindir.

Yük sınıflandırmada ikinci kısım, düğümleri yani her bilgisayarın yük durumunu belirlemektir. Eşik değerleri kullanılarak düğümler beklemede, düşük, normal ve yüksek olarak gruplandırılmaktadır. Her bir düğüm için CPU kullanımı, CPU kuyruk uzunluğu, bellek kullanımı ve ağ trafiği gruplar için kontrol edilir.

$$\boldsymbol{Yük} = \begin{cases} \boldsymbol{Beklemede\ (Idle)} \rightarrow (cpuState == IDLE) \wedge (mem\_u < 70) \\ \boldsymbol{Yüksek\ (High)} \rightarrow ((mem\_u > 85) \wedge (cpuQState == HIGH) \vee \\ \quad (mem\_u > 90) \vee (cpuState == HIGH \wedge memState == HIGH)) \\ \boldsymbol{Düşük\ (Low)} \rightarrow ((cpuState == LOW) \wedge (netState == LOW) \vee \\ \qquad (mem\_u < 70) \vee (cpuqState == LOW)) \\ \boldsymbol{Normal} \rightarrow Aksi\ Halde \end{cases}$$

- Beklemede: Yalnızca CPU durumu, bir düğümün beklemede olup olmadığını belirlemek için yeterli değildir. Bellek kullanım bilgileri de beklemede olan düğümlerin belirlenmesinde kullanılır. Modern ağ sistemlerinde, bazı arka plan yazılımları mevcuttur. Birçoğu belirli kısa aralıklarla çalışmaktadır. Arka planda çalışan programlar için %1'lik bir oran belirlenmiştir. CPU kullanımı %1'e eşit ise veya daha küçük ise ve bellek kullanımı %70 altında ise beklemede olarak belirlenir.
- Yüksek: Eğer aşağıdaki şartlardan bir tanesi doğrulanır ise, düğüm yüksek yük yoğunluğuna sahiptir denebilir.
    - Bellek kullanımı %85'den büyük ise ve CPU kuyruğu ağdaki diğer düğümlere göre daha yüksek oranda iş bekletiyor ise düğüm durumu yüksek olarak belirlenir (Kullanılabilir bellek %15 altında ise bellek sayfalama riski olacaktır. Bu da geçerli düğümün işleme gücünü düşürecektir.).
    - Bellek kullanımı %90'dan daha yüksek ise, düğüm diğer şartlara bakılmadan yüksek olarak işaretlenir.
    - CPU kullanımı ve bellek kullanımı diğer düğümlere göre yüksek oranda ise düğüm yüksek seviyede olarak belirlenir.
- Düşük: Eğer aşağıda verilen iki şart sağlanırsa, düğüm düşük yük yoğunluğuna sahiptir.
    - CPU kullanımı ve ağ kullanımı düşük ise, düşük.
    - Bellek kullanımı %70 altında ise ve CPU kuyruğunda bekleyen iş sayısı düşük seviyede ise düğüm düşük seviyededir.
- Normal: Düğüm yukarıdaki kategorilerde değil ise normal kategoride sayılır.

4.1.2 Yük Dengeleme Sistemi Protokolü (LBS)

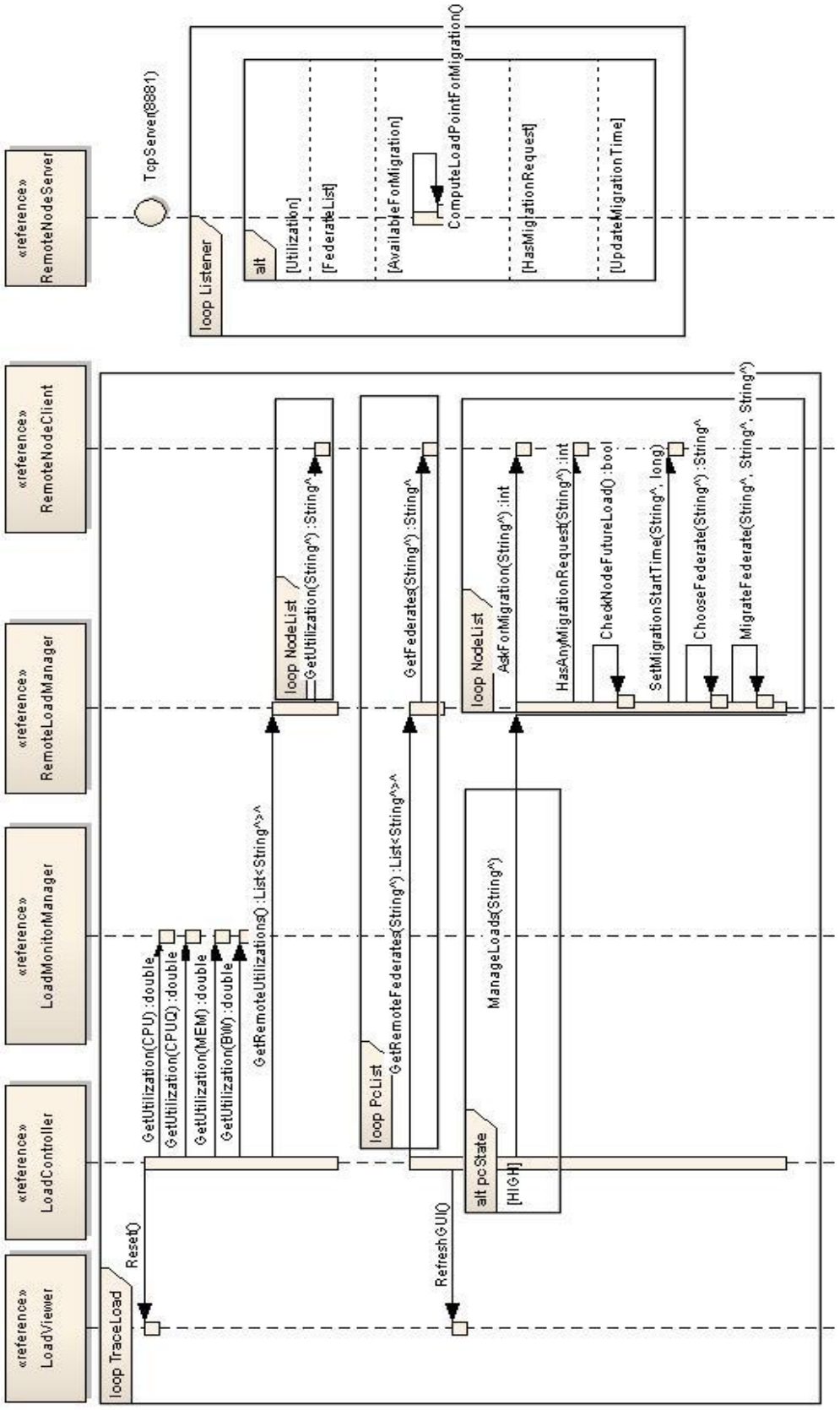
Yük dengeleme sistemi tanımlanmış olan bir dizi prosedürler ile gerçekleştirilmiştir. Bu prosedürler Şekil 4.1'de özetlenmiştir. Buna göre her bir LBS çevriminde

sırasıyla üç temel sorunun cevabı aranır, ne zaman yük dengeleme sürecinin başlatılacağı, hangi düğümler arası yük gönderileceği ve hangi federenin taşınacağı. Bu soruların cevapları sırasıyla şu aşamalar ile gerçekleştirilir;

Yük yöneticisi tarafından, yük izleyicisinin elde ettiği sistem kaynakları, uzak düğümlere ait yük bilgileri ile karşılaştırılır. Bunun için uzak yük yöneticisi üzerinden diğer düğümler ile iletişime geçilerek yük bilgileri elde edilir. Elde edilen yük bilgileri yukarıda verilen yük sınıflandırma algoritması ile sınıflandırılır ve düğümün yük yoğunluğu yüksek olması halinde, yük dengeleme prosedürleri çalıştırılır. Böylelikle ne zaman yük taşınacağı sorusunun cevabı verilmiş olur.

Hangi düğümler arası yükün taşınacağı sorusunun cevabı için, düğümler hakkında güncel yük bilgilerinin elde edilmesi gerekmektedir. Elimizde olan yük bilgileri diğer düğümlerin güncel yük durumlarını göstermeyebilir, bunun için düğümlerin her birine uzak yük yöneticisi aracılığıyla iletişime geçilip durumları hakkında bilgi alınır. Aynı zamanda diğer düğümlere bir aktarım isteği gidip gitmediği düğüm seçiminde bilinmesi gereken bir bilgidir.

Hangi federenin taşınacağı bizim sistemimizde federelerin tahmini yaşam süresine göre belirlenmektedir. Bunun için en uzun yaşam ömrüne sahip federe taşınmak için seçilmektedir.



Şekil 4.1: Yük Dengeleme Sistemi Protokolü

Şekil 4.1'de ki yük dengeleme işlemi her Δt adımında çalıştırılır ve sırasıyla aşağıdaki adımlar gerçekleştirilir;

Yük dengeleme işlemi "LoadController" sınıfı içerisinde başlatılır ve her Δt adımında yürütülür. İlk olarak sistemin adaptif olmasından dolayı her adımda aktif düğüm sayısı belirlenir ve bir liste elde edilir.

Yük dengeleme işlemi başlangıcında kaynak takibi yapılması için "LoadMonitorManager" sınıfı içerisinde bulunan iş parçacıkları başlatmaktadır. Bunlar sırasıyla "CPUMonitor", "MemoryMonitor" ve "NetworkMonitor" sınıflarıdır. Bu sınıflar içerisinde ilgili düğüme ait yük bilgileri anlık olarak tutulmaktadır. Yük dengeleme işleminin ikinci adımında kaynak izleyici sınıflarında tutulan bilgiler toplanmaktadır.

Kendi düğümüne ait bilgiler toplandıktan sonra sırasıyla aktif düğüm üzerindeki bilgisayarlara erişilmektedir ve o bilgisayarlara ait yük bilgileri toplanmaktadır. Bu işlem "RemoteLoadManager" sınıfı üzerinden, "RemoteNodeClient"'a TCP/IP protokolleri ile erişim yapılarak gerçekleştirilmektedir.

Elde edilen kaynak bilgileri ile düğümlerin yük yoğunlukları belirlenmektedir. Eğer ilgili düğümün kendi yük yoğunluğu yüksek ise bir yük dengeleme işlemi başlatılmak istenir. Bunun için düşük yük yoğunluğuna sahip düğümler ile bekleme konumunda olan düğümler arasından seçim yapılması için bir liste oluşturulur ve bu liste "RemoteLoadManager" sınıfına gönderilir.

Bir sonraki adımda "RemoteLoadManager" sınıfı içerisinde, yük durumu düşük olan düğümler arasından bir seçim yapılacaktır. Bu seçim için diğer düğümlerin dinleme konumunda olan "RemoteNodeClient" sınıfına erişilerek bir puan istenir. Bu puan düğümün o anki yük durumu ve üzerinde koşan federelerin yaşam süresi ile hesaplanmaktadır. Elde edilen puanlardan en düşük puana sahip düğüm federe taşınması için en uygun düğüm olarak belirlenmektedir. Puanlar hesaplanırken eğer düğüm yüksek yük yoğunluğuna sahip ise yüksek bir puan verilir, eğer düğüm bekleme konumunda ise en düşük puan 0 verilir diğer durumlarda ise düğümün

üzerindeki federelerin yaşam ömrü puan hesabına dâhil edilerek toplam bir puan elde edilir.

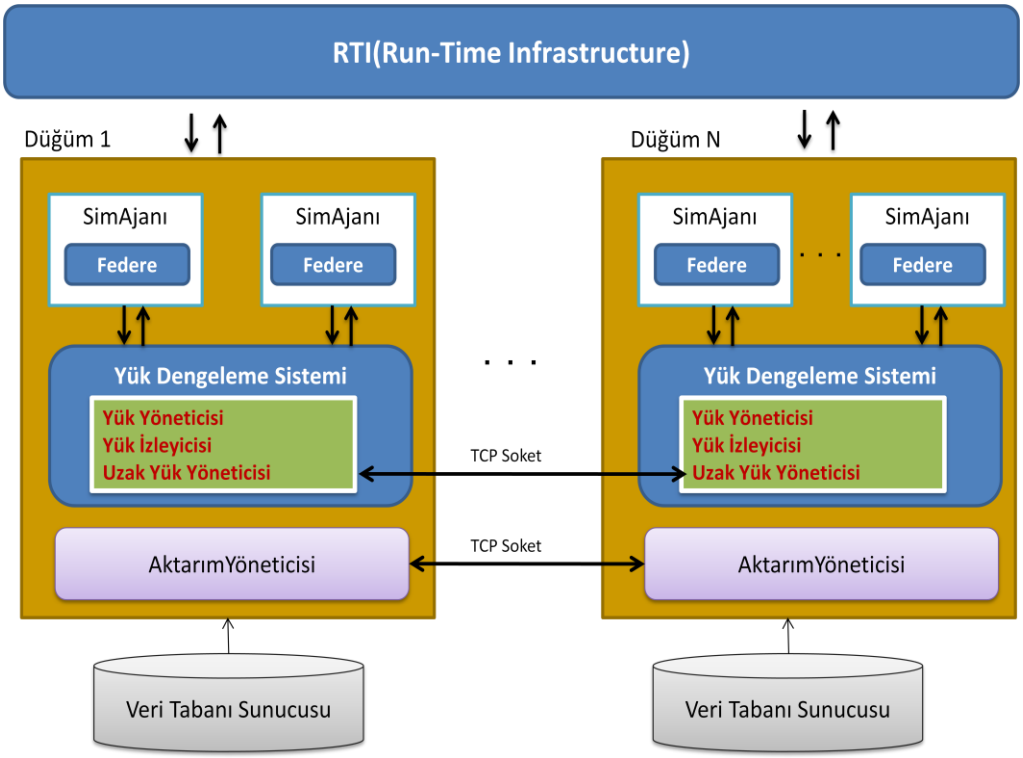
Düğüm seçildikten sonra, hangi federenin taşınması gerektiği federelerin yaşam ömürlerine göre belirlenir ve en uzun simülasyonda kalması muhtemel federe taşınmak üzere seçilir.

Seçilen düğüm ile seçilen federenin kaynak kullanım miktarları karşılaştırılır ve federenin taşınması halinde düğümün bir sonraki döngüde muhtemel yük durumu elde edilir. Yük durumu yüksek ise taşıma gerçekleştirilmez, aksi taktirde düğüm ve federe uygun ise taşıma işlemi gerçekleştirilir.

5 MİMARİ YAKLAŞIM

Geliştirilmiş yük dengeleme sistemi tasarımı Şekil 5.1'de verilmiştir. Tasarlanmış olan sistem genel hatlarıyla üç temel kısımdan oluşmaktadır. Birinci kısım simülasyon koşumlarının yürütüldüğü ve senkronizasyonların sağlandığı RTI alt yapısı, ikinci kısım Bölüm 3'te detaylı şekilde açıklanmış olan simülasyon ajanı ve üçüncü kısım karar verme mekanizması olan ve Bölüm 4'te yer alan yük dengeleme sisteminden oluşmaktadır. Şekil 5.1'de görüldüğü gibi ağda yer alan her bir düğüm içerisinde "SimAjan"'ları ve "Yük Dengeleme Sistemi" yer almıştır. Dinamik yük dengeleme yaklaşımlarında karşılaşılan temel ayrımlardan bir tanesi de karar verme mekanizmasının nerede yer alacağıdır. Merkezi politikalarda her bir düğüme ait bilgiler merkezi bir düğümde tutulur ve bu bilgi kullanılarak merkezi düğümde karar verme mekanizması çalıştırılır. Dağıtık politikalarda ise durum bilgileri her bir düğüme dağıtılır ve her bir düğüm ağdaki diğer düğümlerin durumlarına göre kendi yükünde bir iyileştirmeye gidebilir. Geliştirilen sistemde dağıtık bir yaklaşım benimsenmiştir ve bunun için ağdaki düğümler arası iletişim TCP/IP protokolleri ile sağlanmaktadır. Sistem şekilde görüldüğü gibi birden fazla port üzerinde çalışmaktadır. Bu portlardan bir tanesi aktarım sırasında diğeri ise yük bilgilerinin paylaşılması aşamasında kullanılmaktadır.

Sistem MVC (Model-View-Controller) yazılım mimarisi deseni benimsenerek geliştirilmiştir. Model arayüzü (view), model veriyapısı (model) ve kontrol bileşeni (controller) olarak ayrıştırılmıştır. Model arayüzü, düğümlerin güncel yük durumlarını ve federe sayılarını gösteren kullanıcı grafik arayüzüdür. Model veriyapısı, her bir düğüme ait yük ve federe bilgisinden oluşmaktadır. Kontrol bileşeni ise karar algoritmalarının yer aldığı ve model veri yapısı ile arayüz arasında yer alan orta katmanı oluşturmuştur.



Şekil 5.1: HLA RTI yük dengeleme sistemi genel mimarisi

Şekilde görüldüğü gibi veritabanları her bir düğüme has bilgiler tutmaktadır ve birbirleri hakkındaki bilgileri düğümler ağ üzerinden iletişime geçme metoduyla edinmektedirler, bu da sistemde yer alan düğümlerin tamamen birbirinden bağımsız olması anlamına gelmektedir.

5.1.1 RTI Alt Yapısı

Bu çalışmada ücretsiz olarak kullanılabilen açık kaynaklı HLA RTI uygulaması olan Portico kullanılmaktadır. Portico Java tabanlı geliştirilmiş bir sistemdir, C++ arayüzleri ile C++ uygulamalarında da servisleri kullanıcılara sunmaktadır [7].

6 PERFORMANS ANALİZİ

Geliştirilmiş olan protokol ve algoritmaların performansları aşağıda verilen yöntemler ile test edilmiş ve sonuçları değerlendirilmiştir.

6.1 Test Ortamı ve Sonuçları

Test ortamı, bir dağıtık simülasyon altyapısı oluşturan HLA bağımlı bir uygulama ile saplanmıştır. Üzerinde çalışılacak örnek simülasyon sistemi bir kablosuz sensör ağı (Wireless Sensor Network) simülasyonu olarak düşünülmektedir. Simülasyonda sensörlerin yaşam süreleri ölçülmektedir. HLA federasyonu kablosuz sensör ağı ve üzerinde çalışan sensör federelerinden oluşmaktadır. Her bir sensör birbirlerine belirli aralıklarla rastgele mesajlar göndermekte, tüm federeler aynı mesajı almakta, ancak kendi durumuna uygun olan federe onu kabul etmektedir.

Test, Intel i7-3630QM 4 çekirdekli ve 8 iş parçacığına sahip bir bilgisayar üzerinde yürütülmüştür. Bu bilgisayar üzerinde Oracle VM Virtual Box kurulmuş ve test için sanal makineler oluşturulmuştur. Sanal makinelerin her birine tek çekirdek verilmiş ve testlerin durumuna göre bellek miktarları belirlenmiştir. Her bir bilgisayar üzerinde Microsoft Windows XP Professional SP1 işletim sistemi kurulmuştur. Oluşturulan bu düğümler için yine sanal bir ağ yaratılmıştır ve gerekli HLA konfigürasyonu oluşturulmuştur.

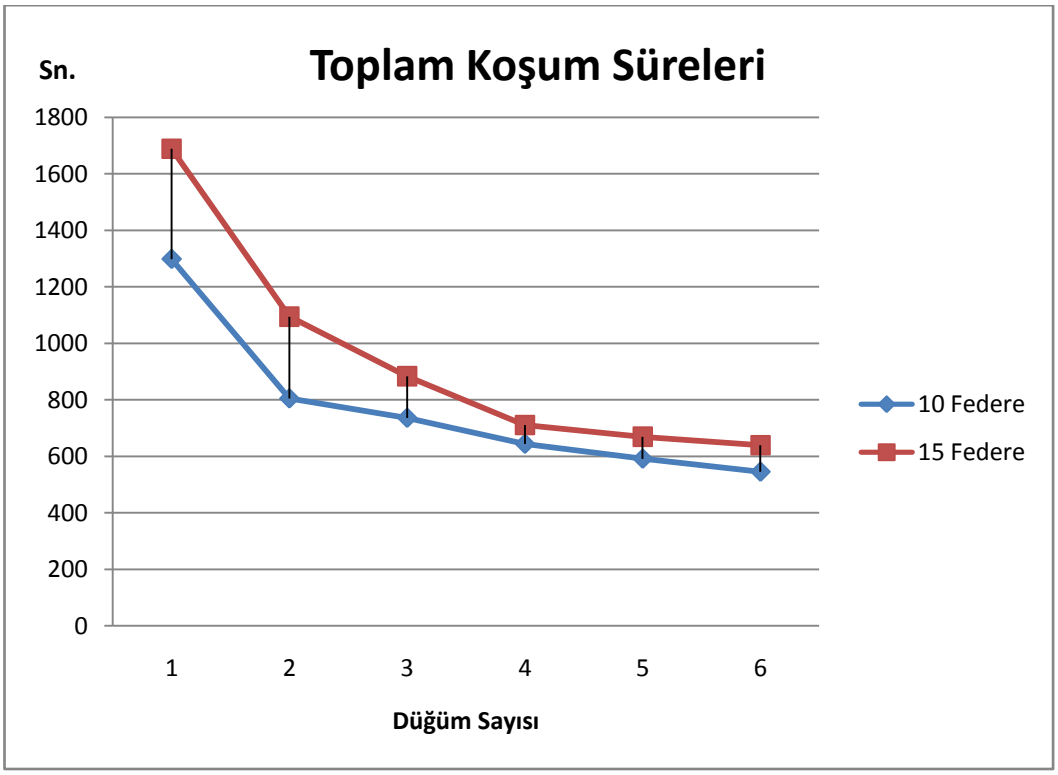
Simülasyon uygulaması;

- Her bir federe, her bir simülasyon adımında sabit işlemci yükü getirmesi için rastgele sayılardan oluşturulmuş bir dizi üzerinde kabarcık sıralama (bubblesort) işlemi yürütmektedir.
- Simülasyonda yer alan federeler 600x400 birim boyutlu bir harita üzerinde rastgele olarak yerleştirilmektedir.

- Rastgele alanlarda oluşturulan federelerin her birinin etki yarıçapı 50 birim olarak belirlenmiştir.
- Federeler arası mesajlaşma sıklığı testlerde verilmiş olan sayı aralıklarında rastgele bir dağılım ile belirlenmektedir.
- Her bir federenin enerji değeri sabit 200 birim olarak belirlenmiştir.
- Federelerin yaşam süresi değişkendir bunun nedeni, her bir federe bir simülasyon döngüsünde 1 birim enerji kaybına uğrar ancak federe mesaj gönderecek veya alacak ise 2 birim enerji harcar.

Yük dengeleme algoritmalarının işleyiş ve etkilerini analiz etmek için bir dizi test yapılmıştır;

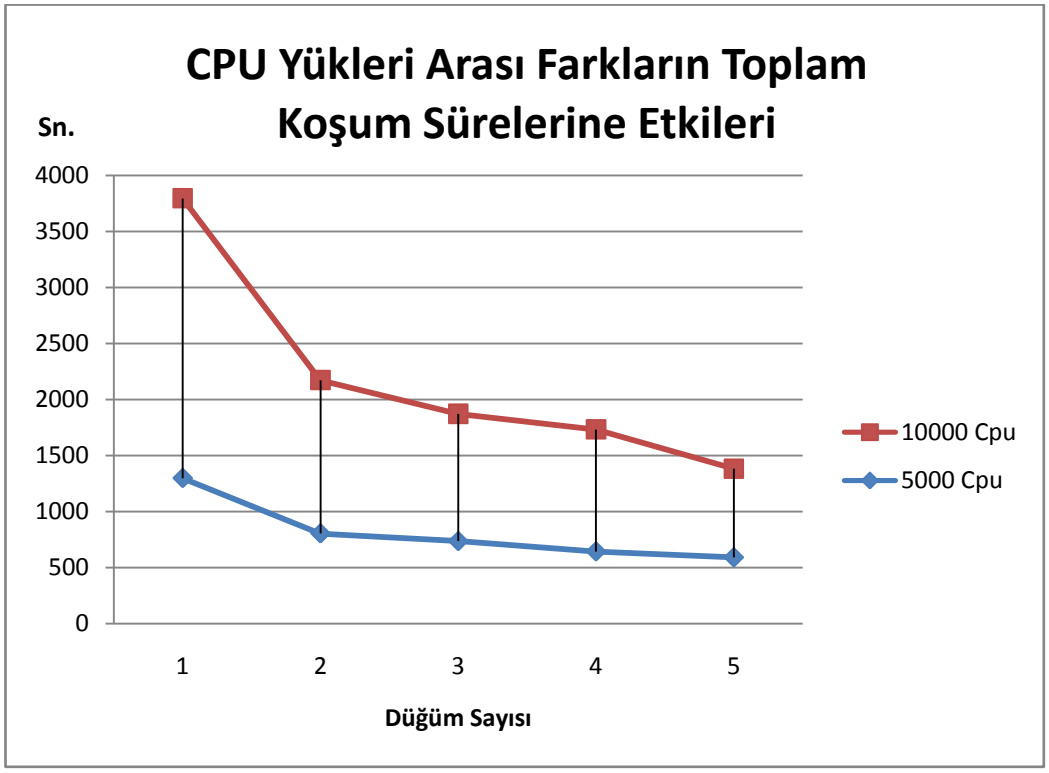
Test 1: Farklı federe sayısı için düğüm sayısının toplam koşum süresine etkisi test edilmiştir. Bu koşum için her bir federe mesaj üretim sıklığı 1-10 sn. arasında bir dağılım ile gönderilmektedir. Her bir düğüm üzerinde işlemci yükünü etkileyen parametre olan eleman sayısı 5000 olarak kullanılmıştır.



Şekil 6.1: Yük Dengeleme Sisteminin Toplam Koşum Sürelerine Etkisi

Şekil 6.1'de görüldüğü gibi sırasıyla 10 ve 15 adet federeye sahip federasyonlar üzerinde koşumlar yapılmış ve koşum süreleri incelenmiştir. Test sonucuna göre tek bir düğüm üzerinde koşturulan federe sayısı arttırıldıkça toplam koşum süresinin de doğru orantılı olarak arttığı gözlenmiştir. Bu durum tek bir düğüm üzerinde ve yük dengeleme sisteminin çalışmadan elde edilmiş sonuçtur. Düğüm sayısı 2'ye yükseltildiğinde toplam koşum sürelerinde yaklaşık %40 oranında azalma olduğu gözlenmiştir. Düğüm sayısı 3'e çıkartıldığında federelerin 2 düğümde çalışmasına oranla %15'lik bir kazanım olduğu gözlenmiştir. Düğüm sayısı arttırıldıkça toplam koşum zamanından elde edilen kazanç ters orantılı bir şekilde azalmaktadır. Bu da sistemin belirli bir düğüm sayısına ulaştıktan sonra elde edilecek kazancın düşeceği ve belkide sisteme ek yük getireceği sonucunu göstermektedir. Ancak yük dengeleme sisteminin devrede olduğu 6 düğümlü koşum ile 1 düğümde yapılan koşum arasında yaklaşık %60'lık bir kazanç sağlandığı gözlenmiştir. Görüldüğü gibi düğüm sayısı arttıkça koşum süresi beklendiği gibi, yük dengeleme algoritmasının yükü dengelemesiyle azalmakta ve federe sayısının etkisi de düğüm başına düşen yükün düşük olmasından ötürü azalmaktadır.

Test 2: İşlemci yük yoğunluğu yüksek bir ortamda sistemin davranışının incelendiği bir takım testler yürütülmüştür. İşlemci yükünün yüksek olması için her bir federe bir simülasyon adımında kabarcık sıralama algoritması çalıştırmaktadır. Bu testlerde kabarcık sıralaması yapılan dizinin boyutu işlemci yükünü doğrudan etkilemektedir. Bunun için ilk testte dizi sayısı 5000 ve ikinci testte 10000 olarak alınmıştır. Bu koşumlar için her bir federe mesaj üretim sıklığı 1-10 sn. arasında bir dağılım ile gönderilmektedir. Mesaj üretim sıklıkları federelerin yaşam sürelerini etkileyen bir parametre olmakla birlikte, federe taşınması sırasında ek yükler getirmektedir.

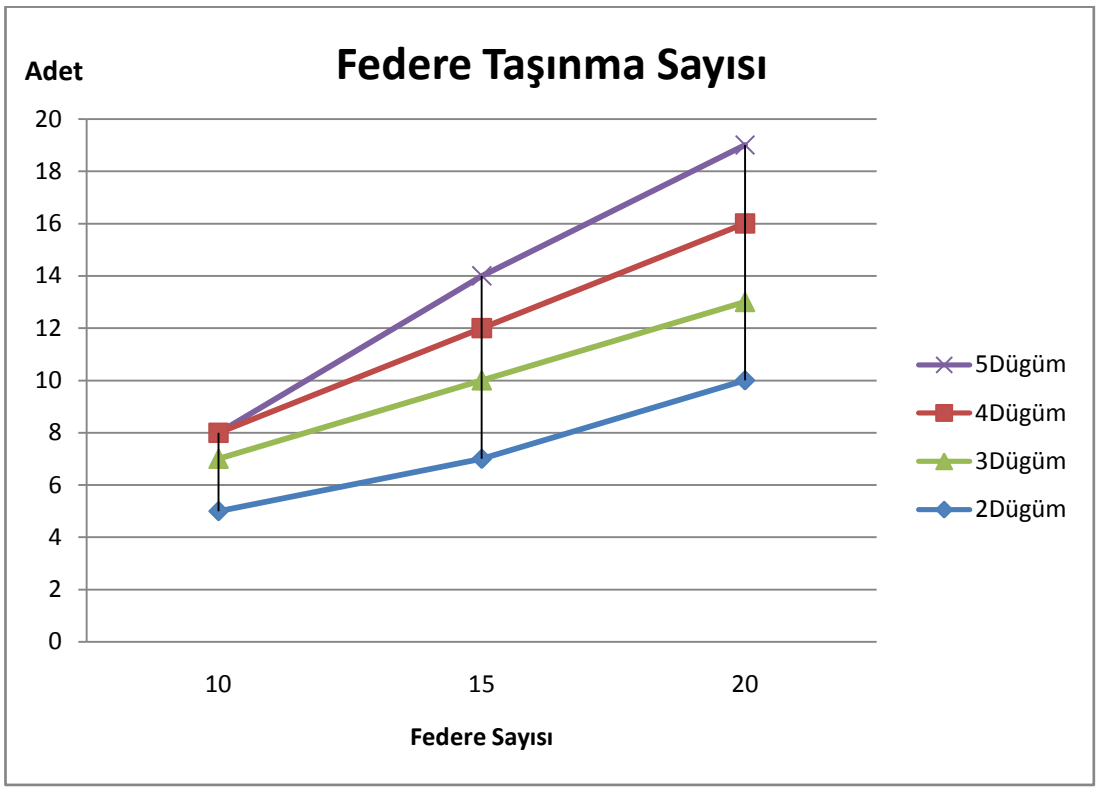


Şekil 6.2: CPU Yükleri Arası Farkların Toplam Koşum Sürelerine Etkileri

Rastgele sayılardan oluşturulmuş 5000 elemanlı ve 10000 elemanlı dizilerde yapılan koşum sonuçları Şekil 6.2'de verilmiştir. Sırasıyla bu testler düğüm sayısı 1'den 5'e kadar çıkartılarak yürütülmüş ve koşum süreleri incelenmiştir. Şekilde görüleceği üzere dizinin eleman sayısı 10000 iken düğüm sayısı 1'den 2'ye çıkartıldığında yaklaşık %60 lık bir kazanım sağlamıştır ancak aynı testler aynı koşullarda 5000 elemana sahip bir dizi ile yapıldığında %40 oranında bir kazanım sağladığı gözlenmiştir. Yani yüksek işlemci yük yoğunluğunda düğüm sayısı arttırıldıkça sistemde meydana gelen kazanımlar da arttığı gözlenmektedir. Son olarak düğüm sayısı 5'e çıkartıldığında yüksek yük yoğunluğuna sahip uygulamada daha fazla kazanım sağlarken buna göre düşük yük yoğunluğuna sahip uygulamada daha az kazanım sağlandığı gözlenmiştir. Bu testler sistemin yüksek işlemci yük yoğunluğunda daha verimli çalıştığını göstermektedir. Bu durum sistemin ölçeklenebilir olduğunun göstergelerinden bir tanesidir. Ayrıca bu testin sonucunda

yüksek işlemci yükü altında federe taşınma süresi ve federe taşınma sayısı sistemin performansını etkilemediği gözlenmiştir.

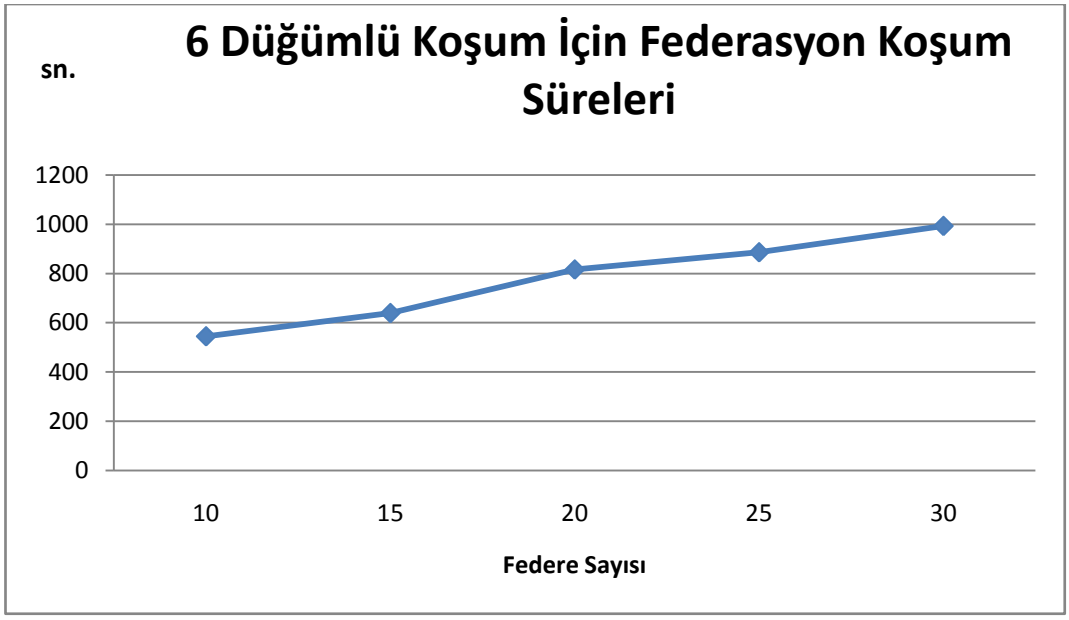
Test 3: Yük dengeleme sistemi devrede iken kaç adet federenin bir düğümden diğer bir düğüme taşındığı gösterilmektedir. Bu testler sırasında düğüm sayıları sırasıyla 2'den 5'e kadar çıkartılmıştır. Düğüm sayısı arttıkça yük dengeleme sırasında taşınan federe sayısı da artmaktadır. Bazı durumlarda, gereksiz federe taşınması sistem performansını kötü etkilemektedir ve toplam koşum süreleri artmaktadır. Koşum performansı, bir federenin düğüme getirdiği yükle bağlantılı olup, yük dengeleme sisteminin sisteme olan kazancını da doğrudan etkilemektedir.



Şekil 6.3: Federe Sayılarına Göre Federe Taşınma Sayısı

Şekil 6.3'de görüldüğü gibi federe taşıma sayıları federasyonda yer alan federe sayısına ve düğüm sayısına bağlı olarak değişmektedir. 10 federe ile yapılan testlerde federe taşınma sayısının değişim aralığının ufak olduğu gözlenmektedir. Federe sayısı arttıkça federe taşıma sayılarınında doğru orantılı bir şekilde arttığı gözlenmektedir. Hatta 5 düğümlü 20 federeye sahip bir koşumda federe taşınmasının neredeyse federe sayısına yaklaştığı gözlenmektedir. Koşumda federeler bir düğüm üzerinde başlatılmakta ve diğer düğümlere yük dengeleme algoritması ile dağıtılmaktadır. Bu yüzden 5 düğümlü 20 federeli koşumda her bir düğümün 4 adet federe yürütmesi ile dengeli bir sistem üzerinde koşum gerçekleştirilmiş olacaktır ve bu koşumda 16 federe taşınması beklenmektedir. Ancak bu koşumda 19 federenin taşınmış olduğu gözlenmiştir, bu durumun sebebi düğümlerdeki bazı federelerin işlerini erken bitirmeleri ile yüklerin diğer düğümlere tekrar dağıtmalarıdır.

Test 4: Yük dengeleme sistemi devrede iken 6 düğümlü bir koşumda sırasıyla 10, 15, 20, 25 ve 30 federe ile koşumlar gerçekleştirilmiştir. Bu testte amaç düğüm sayısı arrtırılmış bir koşumda federe sayılarının toplam koşum süresine etkisi.



Şekil 6.4: 6 Düğümlü Koşum İçin Federasyon Koşum Süreleri

Şekil 6.4'de görüldüğü üzere federe sayısı artmasıyla sistemde meydana gelen kazanım orantılı bir şekilde artmaktadır. 10 federeli koşumda düğüm başına düşen yük miktarı azdır ve yük dengeleme sisteminin devrede olmasıyla sağlanan kazanç ta aynı oranda az olamktadır. Ancak 30 federeli koşumda düğüm başına düşen federe sayısı arttığından dolayı sistemin dengede olması performans açısından önemlidir, bu yüzden dengeli bir sistemde koşumun toplam süresindeki artış federe sayısı arttıkça yük dengelendiği için, göreceli olarak azalmaktadır. Buradan, sistemin genişleme oranına paralel olarak performansta da orantılı iyileşmeler meydana geldiği gözlemlenmektedir.

7 SONUÇ VE GELECEKTEKİ İYİLEŞTİRMELER

Bu çalışmada HLA bazlı simülasyon sistemleri üzerinde federe aktarım yöntemiyle yük dengeleme sistemi geliştirilmiştir. Geliştirilen sistemde herhangi bir üçüncü şahıs yazılım kullanılmamıştır ve aktarımı gerçekleştirebilmek için bırakma (resign), katılma (join), yayınlama (publish) ve abone olma (subscribe) gibi HLA metodları kullanılmıştır. Ayrıca aktarım gecikmesini azaltmak ve simülasyon tutarlılığını sağlamak için, DAFT yani durum aktarımlı federe taşıma yöntemi içerisinde federe kayıt ve tekrar yüklenmesi metodları geliştirilmiştir.

Yük dengeleme sisteminde, yerel ağda simülasyona dâhil olan tüm bilgisayarların yükleri toplanmış ve ağa göre her bir bilgisayara ait yük ağırlıkları çıkartılmıştır. Bilgisayar yüklerinin belirlenmesinde CPU kullanımı, CPU kuyrukta bekleyen iş sayısı, bellek kullanımı ve ağ trafiği unsurları baz alınmıştır. Bu unsurlar ışığında ağdaki yük yoğunluklarının dengelenmesi için doğru kararlar alınması sağlanmıştır.

Sistemin faydasının gösterilmesi için bir takım testler yürütülmüştür. Yapılan yük dengeleme sisteminin toplam koşum süresine etkisi testinde, yük dengeleme sistemi devrede iken çıkan toplam koşum süresi ile devrede değilken çıkan toplam koşum süreleri arasında yaklaşık %60'lık bir kazanç olduğu gözlemlenmektedir. Bu nedenle yük dengeleme sistemi ile entegre çalışan HLA benzetim sistemlerinin daha performanslı çalıştığı ve çalışma sürelerinin kısaldığı gözlemlenmiştir. CPU yükleri arasındaki farkların toplam koşum süresine etkileri testinde sistemin yüksek işlemci yük yoğunluğunda daha verimli çalıştığı gözlenmiştir. Sistem performans ölçütleri düğüm başına düşen federe sayısına bağlı olarak artmakta veya azalmaktadır. Bu yüzden yük dengeleme algortimasının başarımı, yani düğümler arasında yüklerin eşit dağıtılması sistemin performansı açısından önemlidir.

Gelecekte Yapılması Planlanan Geliştirmeler

Sistemin adaptif olması yani uyarlanabilir olması beklenir. Adaptif yaklaşımda, karar verme mekanizması geçmişi, anlık sistem performansını ve geçmişte verilen kararları veya sistemde meydana gelmiş değişimleri de göz önünde bulundurmaktadır. Bu sistemler ayrıca herhangi bir yük parametresinin program performansını etkilememesi durumunda, daha sonraki koşumlarda bu parametrenin geçmişteki ağırlığı göz önünde bulundurularak düşürülmektedir. Adaptif olmayan sistemlerde ise sistemin geçmişinde kullanılan parametrelerin ağırlıkları sürekli aynı kalmaktadır. Geçmişte edinilmiş sistemin karakteristiğini özetleyen bilgiler her bir düğüm tarafından belirli bir veri tabanında loglanarak yük dengelemede karar verme aşamasında kullanılması, sistemin daha geniş donanım konfigürasyonlarında da başarılı sonuçlar vermesi açısından fayda sağlayacaktır.

Bazı durumlarda aktarım yapmak sistem performansını kötü etkilemekte ve aktarım yapılmayan durumlarla neredeyse aynı performans kayıplarına neden olmaktadır. Bu durumun önceden sezilebilmesi için daha önce aktarılmış federelerin aktarım süreleri ve aktarım boyutları loglanarak yük dengelemede karar verilmesi sırasında kullanılması sistemin başarımının arttırılmasını sağlayacaktır.

Şu aşamada yük dengeleme işlemi sistemde bulunan aktif düğümler arasında gerçekleştirilmektedir. Bu durumda 10 federe için 10 düğüm var ise her bir düğüme 1 federe dağıtılmaktadır. Bu da simülasyonun normalden daha uzun bir sürede sonlanmasına sebep olabilmektedir. Bu durumun önüne geçilebilmesi için yük dengeleme sistemi karar verme aşamalarına bir takım ek kontroller eklenerek genişletilmesi gerekmektedir.

Testler sensör ağları federasyonu üzerinde yapılmıştır ancak farklı bir federasyon ve federe uygulaması için de testler yapılmalı, ayrıca farklı federe uygulamaları üzerinde aktarımların sonuçları incelenmelidir.

Sistem en fazla 6 düğüm ve en fazla 30 federe üzerinde test edilmiştir. Bu testlerin daha fazla düğüm ve daha fazla federe ile tekrarlanması gerekmektedir. Testler sanal

makinalar ve sanal ağ üzerinde yapılmıştır. Gerçek bir ağda farklı donanım özelliklerine sahip bilgisayarlar üzerinde sistemin test edilerek, gerçekte nasıl davrandığının incelenmesi gelecekte yapılması planlanan çalışmalara bırakılmıştır.

Dağıtık simülasyonda, HLA RTI uygulaması olan Portico kullanılmaktadır. Sistemin farklı RTI uygulamaları üzerinde çalıştırılması ve performansa etkilerinin gözlenmesi de gelecekte yapılacak geliştirmeler kapsamına alınmıştır.

8 KAYNAKÇA

[1] IEEE Std. 1516, "*IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules*", IEEE, 2000.

[2] IEEE Std.1516.1, "*IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification*", IEEE, 2000.

[3] IEEE Std.1516.2, "*IEEE Standard forModeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template Specification*", IEEE, 2000.

[4] IEEE Std. 1516.3, "*IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP)*", IEEE, 2003.

[5] IEEE web site, "*The world's leading professional association for the advancement of technology*", http://www.ieee.org, Erişim Tarihi: 14.06.2014.

[6] Portico Project, http://www.porticoproject.org, Erişim Tarihi: 14.06.2014.

[7] Portico Project standart HLA 1.3 C++ API, "*http://www.porticoproject.org/index.php?title=Status_CPP13*", Erişim Tarihi: 14.06.2014.

[8] Akçay M., "*Distributed Dynamic Load Balancing For Parallel Computers*". Doktora Tezi, Eskişehir Osmangazi Üniversitesi Fen Bilimleri Enstitüsü, sayfa 12-15, 2008.

[9] Werstein P.,Situ H. and Huang Z., "*Load Balancing in a Cluster Computer*". In Proc. of the Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies, sayfa 343-347, IEEE Computer Society, 2006.

[10] A. Boukercheand R. Grande. "*Optimized Federate Migration forLarge-Scale HLA-Based Simulations*". In Proceedings of the 12th IEEE/ACM International

Symposium on Distributed Simulation and Real-Time Applications. Sayfa 227-235, Ekim 2008.

[11] R. Grande. "*Dynamic Load Balancing Schemes forLarge-Scale HLA-based Simulations*", Pub. Distributed Simulation and Real Time Applications (DS-RT), IEEE/ACM 15th International Symposium, sayfa 4-11, 2011.

[12] Persson A.,"*Migration and Load balancing in HLA based distributed simulations*", Master of Science Thesis Stockholm, Sweden, 2005.

[13] Li, X.,Xiaodong, Z., Yanxia, Q.: "*Effective Load Sharing on Heterogeneous Networks of Workstations*", Proceedings of the fifteenth workshop on Parallel and distributed simulation, sayfa 91-98, 2000.

[14] Johannes Lüthi and SteffenGrBomann. "*The Resource SharingSystem: Dynamic Federate Mapping for HLA-based Distributed Simulation*" Proceedings of the fifteenth workshop on Parallel and distributed simulation, sayfa 91-98, 2001.

[15] Katarzyna Zajac, Marian Buba, Maceij Malawski and Peter Sloot. "*Towards a Grid Management System for HLAbased Interactive Simulations*", InProceedings of theSeventh IEEE International Symposium on Distributed Simulation and Real Time Applications, sayfa 4-11, 2003.

[16] Wentong Cai, Stephen J. Turner and Hanfeng Zhao. "*A Load Management System for running HLA-based simulation over the grid*", In Proceedings of theSixth IEEE International Symposium on Distributed Simulationand Real Time Applications, sayfa 7-14, 2002.

[17] Gary Tan and KC Lim. "*Load Distribution Services in HLA*", InProceedings of the Eighth IEEE International Symposium on Distributed Simulation and Real Time Applications, sayfa 133-141, 2004.

[18] Zijing Yuan, Wentong Cai and Malcolm Yoke Hean Low, "*Federate Migration in HLA-based Distributed Simulation*", In International Conference on Computational Science (ICCS2004), sayfa 87-95, 2004.

[19] AndrewS.Tanenbaum, Maarten Van SteenYuan.,. "*Distributed Systems"*,(2nd ed.).PAERSON, sayfa 17-20, 2006.

[20] NEUMAN, B.: "Scale in Distributed Systems." In Casavant, T. andSinghal, M. (eds.), *Readings in Distributed Computing Systems*, pp. 463-489. Los Alamitos, CA: IEEE ComputerSocietyPress, sayfa 9, 12, 624, 1994.

[21] Marta BELTRÀN ve Jose BOSQUE, "*Information Policies for Load Balancing on Heterogeneous Systems*", IEEE International Symposium on Cluster Computing and the Grid, sayfa 970–979, 2005.

[22] Onur DESTANOĞLU, "*Paralel Algoritmalar İçin Dinamik Yük Dengeleyici Tasarımı ve Uygulamaları*", Yüksek Lisans Tez Çalışması, GYTE Fen Bilimleri Enstitüsü, sayfa 8-23, 2008.

[23] Mark H. WILLEBEEK-LEMAIR ve Anthony P. REEVES, *"Strategies for Dynamic Load Balancing on Highly Parallel Computers"*, IEEE Transactions on parallel and distributed systems, Vol. 4 No. 9, sayfa 979–993, 1993.

[24] Wei JIE, Wentong CAI ve Stephen J. Turner, *"DynamicLoad-Balancing in a Data Parallel Object-Oriented System"*, In Procs. of 2001 International Conference on Parallel and Distributed Systems (ICPADS 2001), sayfa 279-286, 26-29 Haziran 2001.

[25] Wei JIE, Wentong CAI ve Stephen J. Turner, *"DynamicLoad-BalancingUsing Prediction in a Parallel Object-Oriented System"*, Procs of 2001 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2001), Nisan 23-27 2001.

[26] GiuseppeDi FATTA ve Michael R. BERTHOLD, *"Dynamic Load Balancing for the Distributed Mining of Molecular Structures"*, IEEE Transactions on Paralleland Distributed SystemsVol. 17 No. 8, sayfa 773–785, Agustos 2006.

[27] Deyu QI ve Weiwei LIN, "*TGrid: A Next Grid Environment*", First International Multi-Symposiums on Computer and Computational Sciences,sayfa 611–618, 2006.

ÖZGEÇMİŞ

Erdem REŞBER 1987 yılında İstanbul'da doğdu; ilk ve orta öğrenimini aynı şehirde tamamladı. İstanbul Ticaret Odası Anadolu Ticaret Meslek Lisesi Bilgisayar Programcılığı Bölümü'nden 2005 yılında mezun oldu. Aynı yıl İstanbul Ticaret Üniversitesi Bilgisayar Teknolojileri ve Programlama bölümüne başladı ve 2007 yılında iyi derece ile mezun oldu. Beykent Üniversitesi Bilgisayar Mühendisliği bölümünden 2010 yılında mezun oldu ve aynı yıl Agena Bilişim ve Savunma Teknolojileri firmasında savunma sanayisine yönelik simülasyon yazılımı ve modelleme üzerine yazılım mühendisi olarak çalışmaya başladı ve halen aynı yerde çalışmaya devam etmektedir.

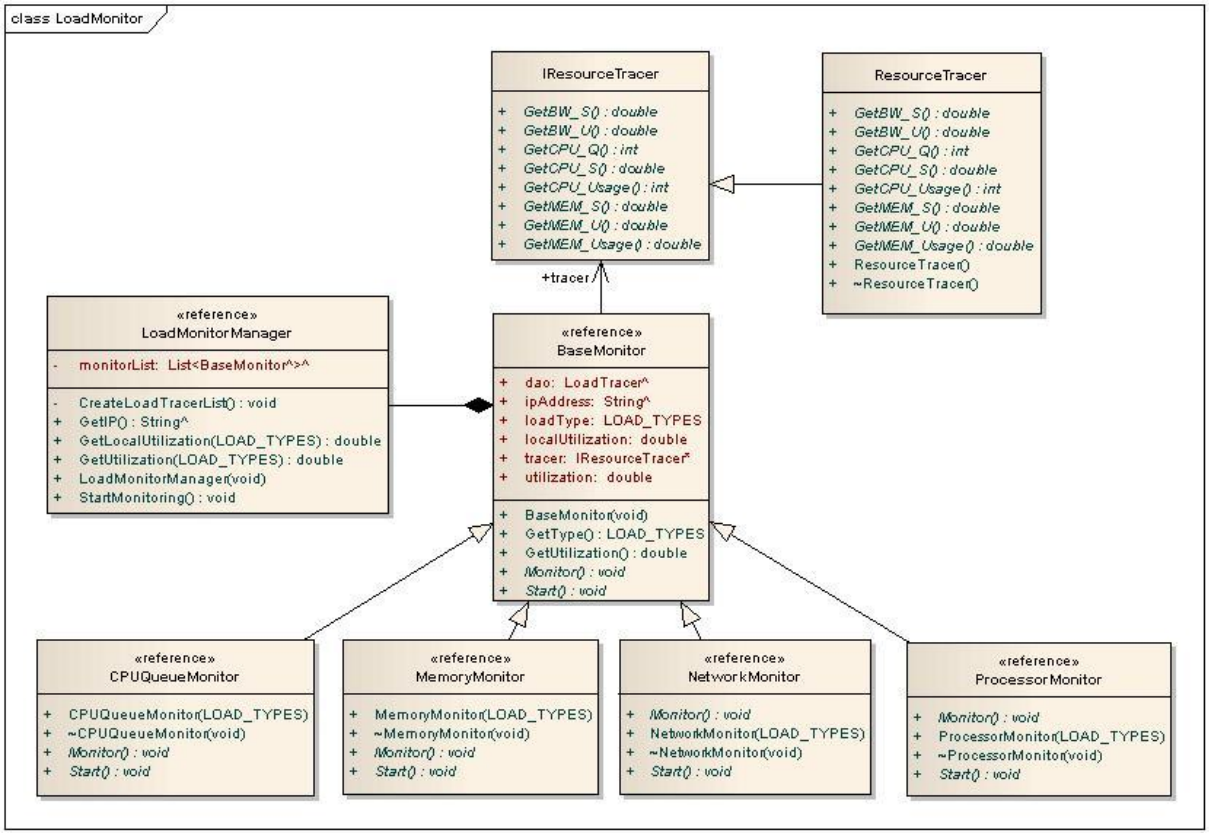
E-posta : erdem_resber@hotmail.com

EKLER

EK-A Yazılım Arayüzleri

1 Yük İzleme (Load Monitor) Paketi

Federelerin koştuğu bilgisayar düğümlerinin, CPU, bellek veya ağ yoğunluğu gibi sistem kaynaklarını izlemek ve bu bilgileri loglamak ile görevli bileşendir. Yük izleyicisi belirli zaman aralıklarında loglama işlemini yürütecektir. Şekil EK-A.1.1'de "Yük İzleme (LoadMonitor)" paketinde bulunan sınıflar ve sınıfları oluşturan metodlar verilmiştir.



Şekil EK-A.1.1: Yük İzleme Paketi Sınıf Diyagramı

Geliştirilen sistem iki temel paketten oluştuğunu ifade etmiştik, "Yük İzleme(Load Monitor)" bunlardan bir tanesi. Yük izleme paketi, "LoadMonitorManager", "BaseMonitor", "ProcessorMonitor", "CPUQueueMonitor", "MemoryMonitor", "NetworkMonitor" ve "ResourceTracer" olmak üzere 7 adet sınıftan meydana gelmektedir. Bu pakette yer alan sınıflar sistem kaynaklarını izlemek ile görevlidir. "BaseMonitor"'den türetilmiş diğer sınıfların her biri ayrı bir iş parçacığı üzerinde kaynak izleme işlemini yürütmektedir. Kaynak izleme sınıfları tarafından elde edilen veriler "LoadMonitorManager(yük izleme yöneticisi)" sınıfı aracılığıyla "LoadManager(yük yöneticisi)"'e iletilir.

1.1 LoadMonitorManager: Yük izleme yönetici sınıfıdır, yük izlemek üzere "MemoryMonitor", "NetworkMonitor", "CPUQueueMonitor" ve "ProcessorMonitor" bu sınıf tarafından başlatılır.

1.2 BaseMonitor: Yük izleyici temel sınıfıdır. "MemoryMonitor", "NetworkMonitor", "CPUQueueMonitor" ve "ProcessorMonitor" bu sınıftan türetilmiştir.

1.3 MemoryMonitor: Bellek yük izleyici sınıfıdır. Bilgisayar bellek bilgileri bu sınıf tarafından izlenmektedir.

1.4 ProcessorMonitor: İşlemci yük izleyici sınıfıdır. Bilgisayar işlemci bilgileri bu sınıf tarafından izlenmektedir.

1.5 MemoryMonitor: Ağ yük izleyici sınıfıdır. Ağ yük bilgileri bu sınıf tarafından izlenmektedir.

1.6 CPUQueueMonitor: İşlemci kuyruğunda bekleyen iş sayısını loglayansınıftır.

1.7 ResourceTracer: Kaynak izleyici sınıfıdır. Bilgisayar kaynakları bu sınıf tarafından toplanarak ilgili sınıflara gönderilmektedir.

2 Yük İzleme (Load Monitor) Paketi

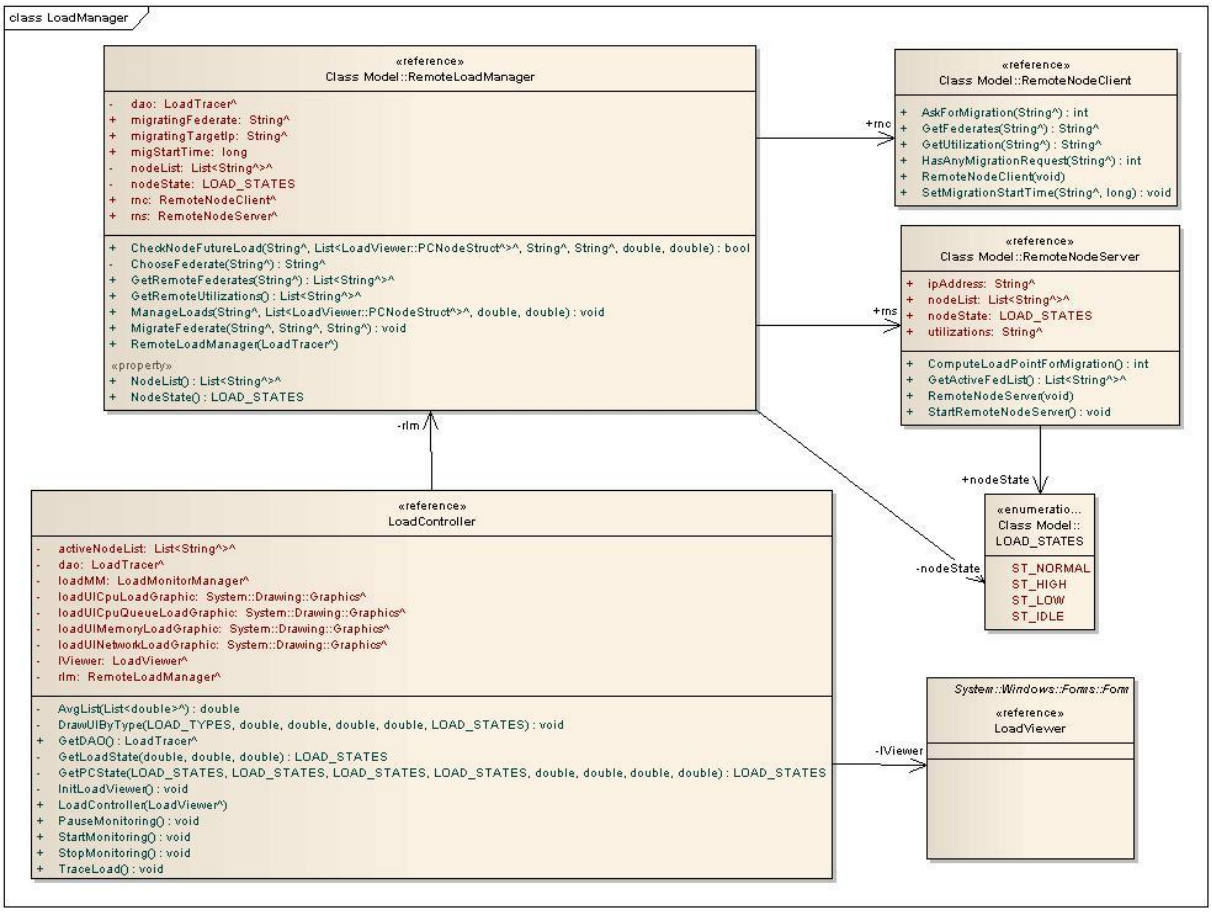
Yük yöneticisi paketi "LoadController", "RemoteLoadManager", "RemoteNodeClient", "RemoteNodeServer" ve "LoadViewer" sınıflarından meydana gelmektedir. "Yük Kontrolcü (LoadController)" sınıfı yük dengesizliklerinin tespitini yapmak ile görevlidir. Bu sınıf yük dengeleme ihtiyacı duyulduğunda "Uzak Yük Yöneticisi (RemoteLoadManager)" sınıfını kullanmaktadır. Uzak düğümler arası bilgi alışverişleri her bir düğümde hizmet alma ve hizmet verme yöntemiyle "Uzak Düğüm İstemcisi (RemoteNodeClient)" ve "Uzak Düğüm Sunucusu (RemoteNodeServer)" sınıfları ile gerçekleştirilmektedir.

2.1 LoadController: Yük dengeleme sistemi kontrol sınıfıdır, yük izleme işlemi bu sınıf tarafından başlatılır. LoadMonitor tarafından elde edilen yük bilgileri ve diğer düğümlere ait yük bilgileri bu sınıfta yük hesabına sokularak düşük ve yüksek eşik değerleri hesaplanır ve bu hesaba göre sistemin yük durumu elde edilir.

2.2 RemoteLoadManager: Uzak yük yöneticisi sınıfının görevi sistemde yer alan düğümler arası bilgi alış verişi yapmak ve ihtiyaç duyulması halinde federe taşınması işlemini başlatmaktır.

2.3 RemoteNodeClient: Uzak düğüm istemcisi ağda yer alan diğer düğümlere erişerek gerekli bilgileri edinmek üzere kullanılan sınıftır.

2.4 RemoteNodeServer: Bu sınıf ağda bulunan diğer düğümlere kendi düğümü hakkında bilgi sağlananhizmete veren sınıftır.



Şekil EK-A.2.1: Yük Yönetici Paketi Sınıf Diyagramı

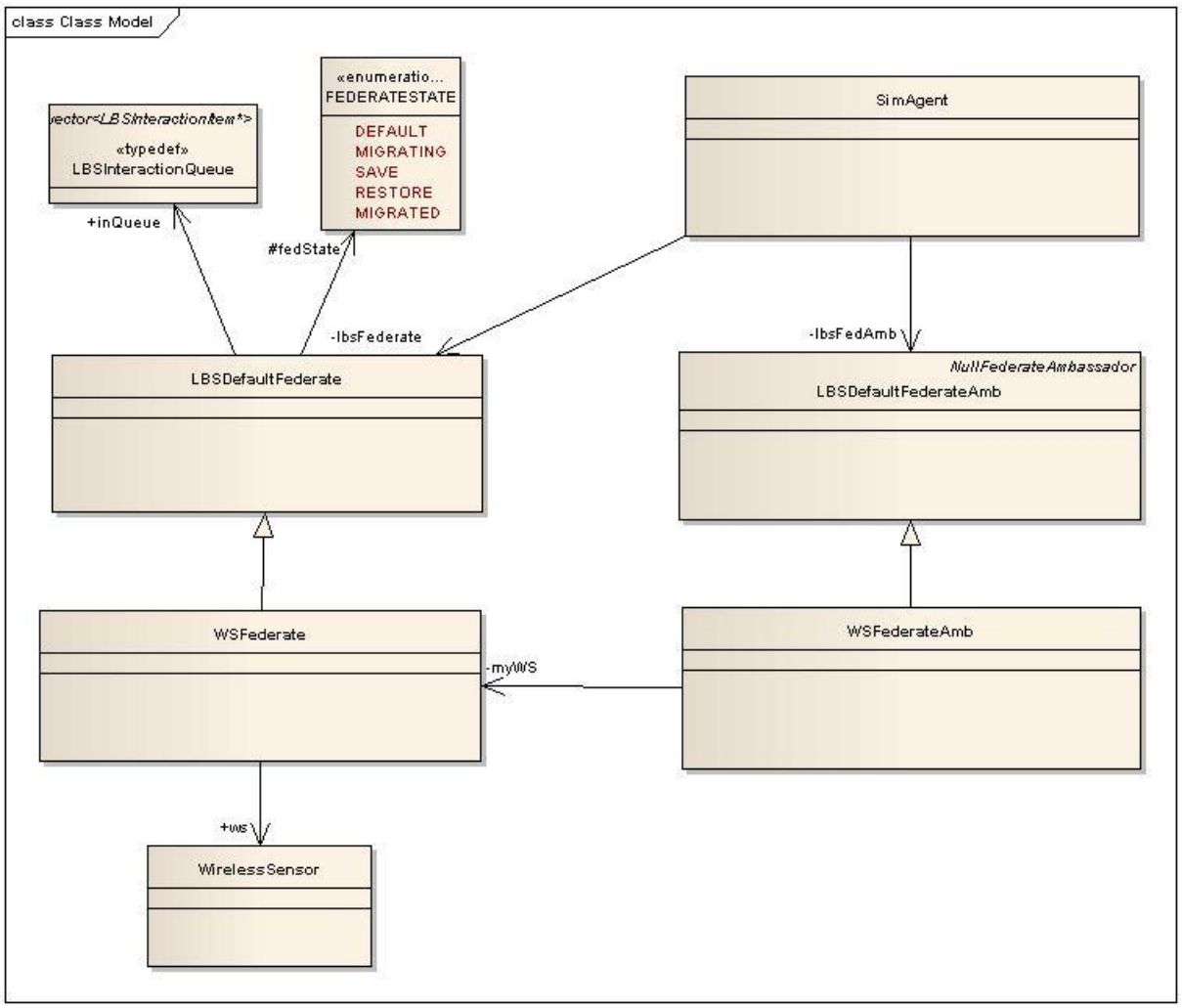
3 Simülasyon Ajanı ve Federe Aktarımı Paketi

Simülasyon ajanı mimarisi bölüm 3.1.1'de detaylı bir şekilde verilmiştir.

LBSDefaultFederate: Federe temel sınıfıdır, yük dengeleme sisteminde uyumlu çalışacak tüm federeler bu sınıftan türetilir.

LBSDefaultFederateAmb: Federe elçisi temel sınıfıdır, federasyonun federe ile etkileşimi bu sınıf üzerinden gerçekleştirilir.

3.1 SimAgent: Simülasyon ajanı yük dengeleme sistemi ile federeler arasında etkileşimi gerçekleştiren sınıftır. Bu sınıf üzerinden yük dengeleme için gerekli servisler(federenin kayıt edilmesi, tekrar yüklenmesi gibi) çağrılır.



Şekil EK-A.3.1: Simülasyon Ajanı ve Federe Mimarisi