**İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ**

# ETMEN TABANLI BİR GRID SİSTEMİNDE CSP YAKLAŞIMI KULLANILARAK GÜÇLÜ TAŞINABİLİRLİKLİ GÖREVLERLE AĞ TRAFİĞİ YÜKÜ DENGELEME

**YÜKSEK LİSANS TEZİ**

**Zafer Altuğ SAYAR**

**Bilgisayar Mühendisliği Anabilim Dalı**

**Bilgisayar Mühendisliği Programı**

**Tez Danışmanı: Prof. Dr. Nadia ERDOĞAN**

**ARALIK 2012**

**İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ**

# ETMEN TABANLI BİR GRID SİSTEMİNDE CSP YAKLAŞIMI KULLANILARAK GÜÇLÜ TAŞINABİLİRLİKLİ GÖREVLERLE AĞ TRAFİĞİ YÜKÜ DENGELEME

**YÜKSEK LİSANS TEZİ**

**Zafer Altuğ SAYAR**
**504081535**

**Bilgisayar Mühendisliği Anabilim Dalı**

**Bilgisayar Mühendisliği Programı**

**Tez Danışmanı: Prof. Dr. Nadia ERDOĞAN**

**ARALIK 2012**

İTÜ, Fen Bilimleri Enstitüsü'nün 504081535 numaralı Yüksek Lisans Öğrencisi **Zafer Altuğ SAYAR,** ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"ETMEN TABANLI BİR GRID SİSTEMİNDE CSP YAKLAŞIMI KULLANILARAK GÜÇLÜ TAŞINABİLİRLİKLİ GÖREVLERLE AĞ TRAFİĞİ YÜKÜ DENGELEME"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

| | | |
|---|---|---|
| **Tez Danışmanı :** | **Prof. Dr. Nadia ERDOĞAN**<br>İstanbul Teknik Üniversitesi | ..... |
| **Jüri Üyeleri :** | **Doç. Dr. Turgay ALTILAR**<br>İstanbul Teknik Üniversitesi | ..... |
| | **Yrd.Doç.Dr.Yunus Emre Selçuk**<br>Yıldız Teknik Üniversitesi | ..... |

**Teslim Tarihi :** **17 Aralık 2012**
**Savunma Tarihi :** **Aralık 2012**

***Anneme ve aileme,***

## ÖNSÖZ

Tez çalışmam süresince yaşadığım tüm sorunlarımda desteğini ve yardımlarını eksik etmeyen, anlayışlı ve çözüm odaklı yaklaşımlarıyla tezimde büyük bir katkısı olan hocam Sayın Prof. Dr. Nadia ERDOĞAN'a teşekkürlerimi sunarım.

Yüksek lisans eğitimim süresince desteklerini esirgemeyen, tezim süresince çalıştığım kurum olan, TÜBİTAK Bilişim Teknolojileri Enstitüsü'ne teşekkür ederim.

Aileme, hayatımın her kesiminde olduğu gibi bu tezi hazırlarken gösterdikleri her türlü destek için teşekkürlerimi sunmayı bir borç bilirim.

Aralık 2012

Zafer Altuğ SAYAR
Bilgisayar Mühendisi

**İÇİNDEKİLER**

**Sayfa**

**KISALTMALAR**

**ACC** : Agent Communication Channel
**ACL** : Agent Communication Language
**AID** : Agent Identifier
**AMS** : Agent Management System
**CPU** : Central Processing Unit
**CSP** : Constraint Satisfaction Problem
**DF** : Directory Facilitator
**FIPA** : Foundation for Intelligent Physical Agents
**JACOP** : Java Constraint Programming solver
**JADE** : Java Agent Development Toolkit
**JVM** : Java Virtual Machine
**KQML** : Knowledge Query Manipulation Language
**RMI** : Remote Method Invocation
**LPI** : Last Performed Instruction
**MK** : Mesajlaşma Katsayısı
**ÖK** : Özellik Katsayısı
**WCSP** : Weighted Constraint Satisfaction Problem

## ÇİZELGE LİSTESİ

**Sayfa**

## ŞEKİL LİSTESİ

**Sayfa**

# ETMEN TABANLI BİR GRID SİSTEMİNDE CSP YAKLAŞIMI KULLANILARAK GÜÇLÜ TAŞINABİLİRLİKLİ GÖREVLERLE AĞ TRAFİĞİ YÜKÜ DENGELEME

## ÖZET

Günümüzde düşük maaliyetli ve yüksek hızlı bilgisayarların çoğalmasına ek olarak çok hızlı ve geniş bantlı bağlantı olanaklarının yaygınlaşması ile bilgisayarlar arası kaynak paylaşımına dayalı sistemler oldukça popüler bir hal almıştır. Grid sistemleri fiziksel olarak farklı tiplerde olan ve bir ağ üzerinde farklı konumlarda bulunan kaynakların, dağıtılmasını ve bu kaynaklara erişimlerinin koordine edilmesini sağlayan dağıtık sistemlerdir. Yazılım gridleri kullanılarak oluşturulabilecek sanal organizasyonlar sayesinde süper bilgisayarlar seviyesinde işlemci kapasitesine sahip sistemler oldukça düşük maaliyetlerle oluşturulabilmektedir. Elde edilen işlem gücü ile ticari ve bilimsel alanlarda birçok karmaşık hesaplama makul sürelerde yapılabilmektedir. Dağıtık hesaplamanın başka bir dalı olan etmen sistemleri yapay zeka ile dağıtık sistemlerin esnek bir birleşimidir. Etmen sistemleri günümüzde ticari ve bilimsel birçok uygulama alanında üzerinde sıklıkla çalışılan bir bilgisayar bilimleri konusu olarak göze çarpmaktadır.

Dağıtık sistemlerin kullanım alanlarının artması ve birbirinden bağımsız bir çok kullanıcının dağıtık sistemlerin sunduğu kaynak ve hizmetlerden yoğun olarak faydalanmasıyla birlikte çeşitli yük dengeleme mekanizmaları dağıtık sistemler için oldukça önemli hale gelmiştir. Özellikle gridler gibi soyut ve gevşek bağlı ilişkilerin yoğun olduğu sistemlerde, kullanıcıların sunulan hizmetin veya dağıtılan kaynağın sürekli bir şekilde kaliteli olarak sürdürülmesi beklentilerinin karşılanması gereksinimi, bu tarz sistemlerde yük dengeleme ihtiyacını arttırmaktadır.

Taşınabilirlik bir uygulamanın çalıştığı bir sistemden alınıp çalışmasına başka bir sistemde devam etmesinin sağlanmasıdır. Taşınabilirlik sayesinde dağıtık sistemlerde yük dengeleme, ağ iletişiminin azaltılması ve sistem sürekliliğinin sağlanması gibi işlemlerin yapılması mümkündür. Temelde güçlü ve zayıf taşınabilirlik olmak üzere iki çeşit taşınabilirlik vardır. Güçlü taşınabilirlikli sistemlerde taşınan uygulama duraklatılarak hedef sistemde çalışmasını kaldığı yerden devam ettirilir. Zayıf taşınabilirlikte ise taşınan uygulama hedef sistemde çalışmasına baştan başlar.

CSP'ler yapay zekanın çeşitli problemlerinin formüle edilmesinde ve çözülmesinde kullanılan yaygın bir yöntemdir. CSP yani kısıt karşılama problemleri belirli bir problemle ilgili tüm kısıt ve limit tanımlamaları yapıldıktan sonra bu kısıtları sağlayacak bir çözüm kümesini arama işlemi olarak tarif edilebilir. Ağırlıklı CSP'lerde(WCSP) ise tanımlanan kısıtlar için ağırlık değerleri belirlenebilmektedir. Bu tarz kısıtlar tanımlayabilmek bize problemimizde bir kısıtın gerçekleşmesi durumunda sisteme getirebileceği maliyeti tanımlayabilme ve bu maliyetin en düşük olacağı çözümü arama imkânı vermektedir

Tezimizde etmen tabanlı bir grid sistemine bir ağ trafiği yükü dengeleme

mekanizması önerilmiştir. Bu kapsamda FIPA standartlarına uygun şekilde tasarlanmış etmenlerden oluşan bir etmen tabanlı grid sistemine, gridde çalıştırılan uygulamaların(görevlerin) güçlü taşınabilirlikli bir şekilde taşınarak çalışma zamanında ağ trafiği yükü dengelemesi yapabileceği mekanizmalar eklenmiştir. İlgili mekanizmalar gerçeklenirken, kullanılan etmen tabanlı grid sisteminin yaşam döngüsünde ve sunduğu mesajlaşma altyapısında değişiklikler ve bunlara çeşitli eklemeler yapılmıştır. Etmen tabanlı grid sisteminde güçlü taşınabilirlik gerçeklenirken Java dilinde geliştirilmiş bir iplik serileştirme kütüphanesi kullanılmıştır. Güçlü taşınabilirlikli görev göçünün sağlanmasında kullanılan serileştirme mekanizmalarına ait görev yürütücüsünün etmen tabanlı grid sistemimize entegrasyonu sırasında karşılaşılan sorunlar safha safha çözümlenerek sistem çalışır hale getirilmiştir. Güçlü taşınabilirlik ve mesajlaşma altyapılarına yapılan eklentilerle birlikte, etmen tabanlı grid sistemi çalışma zamanında yük dengeleme yapılabilecek bir uygulama alanı haline getirilmiştir.

Tezimizin asıl önerisi olarak sunabileceğimiz yapı ise etmen tabanlı grid sistemine ait ağ trafiği dengeleme mekanizmasının bir ağırlıklı CSP(kısıt karşılama problemi) şeklinde formulize edilerek gerçeklenmesidir. Tezimizde ağ trafiği yükü dengeleme algoritmamız grid sistemi açısından yük olarak belirlediğimiz çeşitli ağırlıkları olan kısıtların oluşturduğu bir CSP'nin çözümü olarak gerçeklenmiştir. Gerçeklenen bu mekanizma sayesinde sistemimiz açısından yük olarak gördüğümüz yeni bir grid parametresinin uygun formatta CSP'ye ağırlığıyla birlikte kısıt olarak eklenmesiyle ağ trafiği yükünü dengeleme algoritmamız yeni belirlediğimiz yüklere göre karar verir hale getirilebilmektedir. Bunlara ek olarak tezimizde kullandığımız tüm geliştirme araçları ve kütüphaneler Java dili tabanlı olduğundan, geliştirdiğimiz etmen tabanlı grid sisteminde ağ trafiği dengeleme işlemi Java dilinin sağlamış olduğu platform bağımsız çalışabilme özelliğine uyumlu bir yapı arz etmektedir. Sonuç olarak verdiği kararlar bir CSP'ye kısıtlar ekleyerek değiştirilebilecek esneklikte, platform bağımsızlık konusunda desteği olan bir ağ trafik yükü dengeleme algoritması ortaya çıkmıştır.

## NETWORK LOAD BALANCING WITH STRONG MIGRATION IN AN AGENT BASED GRID SYSTEM USING CSP APPROACH

## SUMMARY

Grid computing and multi agent systems are two popular research topics. Nowadays the combination of two topics is in an increasing trend. This thesis combines these topics via presenting a runtime dynamic task load balancing approach between software agents which are running on a grid system.

Grids are hardware and software infrastructures which are new way of the parallel and distributed computing. Grids support the distribution, sharing and coordinated use of heterogeneous resources . The systems which are connected by a grid might be in the same location or distributed globally, running on different hardware or operating systems, and owned by different organizations.

Agents are flexible, encapsulated and autonomous software components, which execute an assigned task by interacting with other components in the environment. Agents are also social components, which are able to make communication and coordination with other components in the environment. They can collaborate on an execution of a task or make collaboration to achieve a total goal. With the aid of heterogeneousness, coordination and distribution power of grids, agents become more efficient and adaptive solutions for variety types of tasks.

In grids execution performance and efficiency are generally important expectations of users. Grids use effective load balancing algorithms for distribution of tasks to overcome these expectations. A load balancing algorithm aims to improve the total utilization of available resources and efficient assignment of tasks in the grid. Load balancing algorithms can be classified into two categories: static or dynamic. In static algorithms, load balance decisions making mechanisms are defined at compile time when resource requirements are estimated. A multicomputer with dynamic load balancing allocate/reallocate resources at runtime based on no a priori task information, which may determine when and whose tasks can be migrated.

Process migration via strong mobility is an efficient mechanism to achieve dynamic load balancing. Strong mobility allows the transfer of an executing process from a source site to a distant site, where it resumes the execution from the interruption point. Strong mobility can be provided by various mechanisms. These mechanisms can be generally implemented in four levels of programming concept, these are operating system level, interpreter level, source code level, compiled code level. Each level has trade-offs between efficacity and portability In this thesis we used compiled code level approach which is implemented in Java. The portability of this technique is independent from the operating system and the JVM.

CSP(Constraint Satisfaction Problem) is a problem composed of a finite set of variables each of which has a finite domain of values and a set of constraints. Each

constraint is defined over some subset of the original set of variables and restricts the values these variables can simultaneously take. The task is to find an assignment of a value for each variable such that the assignments satisfy all the constraints. The Weighted Constraint Satisfaction Problem (WCSP) is a well known soft constraint framework for modeling over-constrained problems with practical applications in domains such as resource allocation, combinatorial auctions and bioinformatics. WCSP is an optimization version of the CSP framework in which constraints are extended by associating costs to tuples.

In this thesis we implemented dynamic load balancing via strong migration over an agent-based grid system for sharing of processing power resources and task executing on remote platforms. So we extended the agent based grid environment by adding strong mobility and dynamic load balancing features. We used a byte code transformation library written in Java to achieve strong mobility of tasks. We formulated the loads of the grid as weighted CSP constraints. Thus the load balancing decision problem of grid becomes a weighted CSP solving problem. We used a generic CSP solver library which is implemented in Java to solve the WCSP.

We used a grid system to share processing power and to execute tasks on remote platforms AGrid. The AGrid system is implemented on JADE (Java Agent Development Framework) which is developed by Telecom Italia SpA. JADE is a distributed runtime environment on which mobile agents can live, communicate and run parallel tasks via behaviors. JADE also supports graphical user interfaces that can be used for debugging, monitoring, logging and management of the agent system. In addition, JADE is compliant with FIPA specifications, which enables the agents to communicate and cooperate with other agent systems which are also compliant with the FIPA standards. The network traffic load is determined as main load of the grid system, which will be balanced. So we extended AGrid with mechanisms that allows the agents to provide the tasks communicating with each other and enables the tasks to migrate over machines on the grid.

We assumed that each container works on physically different machine in the grid system. The migration analysis algorithm aims to reduce the network traffic load via migrating the frequently messaging tasks to the same container/machine. We used the Jacop constraint solver to implement the migration algorithm. We formulated the loads of the grid as weighted CSP constraints and imposed each load of the system as constraints to the solver. The migration analysis algorithm is generated as follows. Firstly we create a store object to impose all the loads/constraints onto using Jacop. Then we create our CSP variables which represent the tasks of our grid system. We assign the values to the variables which can take in the CSP search operation. The variables are the containers (machines). The true value-variable (task-machine) matching operation which will reduce the inter-machine messaging is our expectation from the CSP solving calculation. So we impose the inter-machine messaging count of each task, which the information is gathered from workers to the migration manager during task executions, as a cost for calculation using weighted constraints. So increasing of inter-machine messaging count of tasks increases the total cost at the same rate. Then we add the possible migrations of tasks to another machine to the calculation as another weighted constraint. Namely if an agent moves on another machine the system takes a task moving cost and adds it to the total cost function. Then we impose the strict relations of tasks to machines as constraints. For example if a task needs to a specific resource, the algorithm constraints that task to the machines which supplies that resource. Thus the load balancing decision problem

of grid becomes a weighted CSP solving problem. We used a generic CSP solver library called Jacop which is implemented in Java to solve our migration analysis problem as WCSP. After execution of CSP problem Jacop generates value-variable assignments. These assignments are task-container (machine) matching which will reduce inter-container messaging count for our grid system. Then the grid applies the decisions of Jacop calculation on grid.

Strong mobility can be provided by various mechanisms. The approach we used is compile code level which let us to work on standard JVMs. To be able to migrate a thread, its execution must be suspended and its stack and program counter must be captured in a serializable format which is then going to be sent to the target location. We used the Brakes library to implement the strong mobility. Brakes has a bytecode transformer which instruments Java classfiles so they can capture their internal states at any time. Basically Brakes can rewrite a Java classfile with including computation state information. That allows us to pause and resume Java threads whenever desirable. We used Brakes thread serialization technique to apply compiled code level strong mobility approach. All the tasks which is run on grid is rewrote at compiled code level to achieve strong mobility.

The migration mechanism works as follows: Firstly the migration manager report the migratory worker agent of task to the migration acceptor worker agent which run the task that will be migrated. Secondly migratory worker captures the task execution with Brakes. Then migratory worker serializes the task and execution state with standard Java serialization. After that migratory worker sends the task to the migration acceptor worker which is assigned by the migration manager. Finally migration acceptor worker deserializes the task and resumes it with Brakes. The migration manager informs the worker agents after migration processes about changes of the locations of the migrated tasks.

The migration mechanism is designed as having the ability of working highly seperate from grid execution. And we have also implemented a test tool allows us to create virtual grid environments which have tasks that are messaging each other. This provides us to test the migration algorithm separately from grid environment execution by generating virtual environments. The test tool can emulate data which our migration algorithm needs from grid to perform migration analysis. The migration algorithm can solve the real world grid data and the artificial grid data which is generated from test tool with the same way. So we applied two types of tests in this thesis, one of them is creating a real grid which has real tasks that have resource needing, realizes messaging of tasks with each other, contains multiple real machines and achieves real task migrations from a machine to another. The other test type is emulating a big sized grid environment data, which has virtual machines, messaging of tasks, resource needing of tasks, count of workers and clients. Both test types the algorithm makes decisions the same way.

In this thesis we used weighted CSP approach for migration analysis on a grid environment. We presented a dynamic solution to implement different migration decisions. We can change easily the migration analysis algorithm by adding new loads for the grid system as constraints to the CSP problem. Adding new constraints changes the migration analysis algorithm and the migration decisions totally. Therefore the load balancing algorithm which we presented is so flexible for balancing of different purposes. For example if we add the processing power of each machine as a load for the grid, the algorithm will work as processing load balancer.

All grid components, algorithms and the applied migration approach are fully implemented in Java and the grid system can work over standard JVMs. This makes our approach platform independent.

# 1. GİRİŞ

Bilgisayar bilimlerinde dağıtık sistemler ve dağıtılmış hesaplamalar önerildikleri günden itibaren önemleri ve popülerlikleri giderek artmış olan konulardır. Günümüzde düşük maliyetli ve yüksek hızlı bilgisayarların çoğalmasına ek olarak çok hızlı ve geniş bantlı bağlantı olanaklarının da artmasıyla dağıtılmış sistemler ve dağıtık hesaplamaların kullanım alanları ve önemleri daha da artmıştır. Dağıtık sistemlerde öne çıkan konulardan bir tanesi de taşınabilirliktir. Taşınabilirlik sayesinde dağıtık sistemlerde yük dengeleme, ağ iletişim trafiğinin azaltılması ve sistem sürekliliğinin arttırılması gibi işlemler yapılabilmektedir. Bu tezde önerilen yaklaşım, etmen tabanlı bir grid yapısı üzerinde etmenler tarafından yürütülen görevlere güçlü taşınabilirlik özelliğinin eklenmesiyle, grid sisteminin ağ yükünün azaltılması amaçlamaktadır. [1]

Grid sistemleri fiziksel olarak çeşitli tiplerde olan ve farklı konumlarda bulunan kaynakların ortaklaşa kullanılmasını, dağıtılmasını ve erişiminin koordine edilmesini sağlayan dağıtık sistemlerdir [2]. Etmenler belirlenmiş bir görevi yerine getirmek üzere fiziksel olarak farklı konumlara dağılmış, çalıştırıldığı ortamda çevresindeki sistemlerle esnek ve otonom ilişkiler kurabilen akıllı yazılımlardır [3]. Taşınabilirlik bir uygulamanın çalıştığı sistemden alınıp çalışmasına başka bir sistemde devam etmesinin sağlanmasıdır. Temelde güçlü ve zayıf taşınabilirlik olmak üzere iki çeşit taşınabilirlik vardır. Güçlü taşınabilirlikli sistemlerde taşınan uygulama duraklatılarak hedef sistemde çalışmasını kaldığı yerden devam ettirilir. Zayıf taşınabilirlikte ise taşınan uygulama hedef sistemde çalışmasına baştan başlar. Önerilen tezde güçlü taşınabilirlikli görevler kullanılarak etmen tabanlı bir grid sisteminde ağ trafik yükünün azaltılması hedeflenmiştir. Tezin ilerleyen kısımlarında güçlü taşınabilirlilik kavramı yerine aynı anlamı ifade etmek üzere göç etme tabiri kullanılacaktır.

Gridler yazılımsal bir ortamdaki her türlü kaynağı kullanıma sunacak şekilde tasarlanabilirler. Tezimizde kullanılan etmen tabanlı grid işlem gücü paylaşımı yapmaktadır. Etmenler tarafından yürütülen görevler işlem gücü kaynağını

kullanmaktadır. Etmenler birbirleriyle etkileşim içinde olan sosyal ve dinamik yazılım bileşenleridir. Tezimizde kullandığımız etmenler çalıştırdıkları görevlerin birbirleriyle mesajlaşabilmelerini sağlamaktadırlar. Tezimizin önemli bir kısmı yoğun şekilde mesajlaşan görevlerin çeşitli bağımlılıklarına göre grid üzerinde uygun etmenlere taşınması işlemidir. Etkili bir görev taşınma(göç) kararının nasıl bir yolla verileceği burada önem kazanmaktadır. Bu tezde göç etmesi halinde ağ trafik yükünü hafifletecek olan görevlerin belirlenmesi, göç işleminin hangi etmenden hangisine gerçekleştirileceğinin belirlenmesi işlemi CSP (Constraint Satisfaction Problem) şeklinde formüle edilerek çözülmüştür.

CSP'ler yapay zekanın çeşitli problemlerinin formüle edilmesinde ve çözülmesinde kullanılan yaygın bir yöntemdir. CSP yani kısıt karşılama problemleri belirli bir problemle ilgili tüm kısıt ve limit tanımlamaları yapıldıktan sonra bu kısıtları sağlayacak bir çözüm kümesini arama işlemi olarak tarif edilebilir. Tezimizde görevlerin uzak etmenler tarafından çalıştırılan görevlerle mesajlaşma sayıları, grid üzerindeki etmenlerin bölgesel dağılımları gibi veriler göç karar sistemin kısıtları olarak belirlenmiştir. Bunlara benzer şekilde daha sonradan belirlenebilecek yeni bir kısıtın eklenmesine imkan sağlanarak esnek ve etkili bir göç karar algoritması tasarlanmıştır. Tanımlanan kısıtların çözülmesinde açık kaynaklı bir kısıt çözüm motoru kullanılmıştır.

Tezimiz kapsamında beklentimiz etmen tabanlı bir grid sisteminde ağ trafiği dengeleme uygulamasıyla, farklı bilgisayarlarda yer alıp birbirleriyle yoğun şekilde mesajlaşan görevlerin aynı bilgisayarlara taşınması sonucu, farklı makineler arasındaki mesajlaşma sayılarının azaltılarak ağ yükünün hafiflediğinin gözlemlenmesidir.

### 1.1 Tezin Bölümleri

Tezde öncelikle kullanılan etmen tabanlı grid sistemi tanıtılmıştır. Daha sonra güçlü taşınabilirlikli görev göçünün çalışma mekanizmasından bahsedilmiştir. Sonraki bölümde göçe karar verme algoritması anlatılmıştır. Son olarak da gerçeklenen sistemin ayrıntılarına yer verilerek, yapılan test sonuçları, sistemle ilgili gelecek planları ve önerilen sistemin kullanım alanları tartışılmıştır.

Bölüm 2'de etmen sistemleri ve grid sistemlerine ile ilgili genel bilgiler verilmiştir. Daha sonra kullanılan etmen tabanlı grid sisteminin mimarisi ve özellikleri

anlatılmıştır. Tez kapsamında kullanılan etmen tabanlı grid sistemine yapılan eklentiler de bu bölümde ele alınmıştır.

Bölüm 3'te güçlü taşınabilirlikli görev göçünün Java dili kullanılarak gerçeklenme yollarından bahsedilmiştir. Bu amaçla tezimizde kullandığımız Brakes [4] kütüphanesi tanıtılmış ve ayrıntılı olarak ele alınmıştır.

Bölüm 4'te grid sistemlerinde yük dengeleme konusuna değinilmiştir. Daha sonra tezimizde kullanılan grid üzerinde gerçekleştirdiğimiz ağ trafik yükü dengeleme algoritmasından, bir görevin göç edeceği hedef makinenin seçimine karar veren yapıdan bahsedilmiştir. Bu bağlamda kısıt karşılama problemi (CSP) şeklinde formüle edilen problemlerin çözümünde kullanılan Java tabanlı Jacop [5] kütüphanesi tanıtılmıştır.

Bölüm 5'te önerilen güçlü taşınabilirlikli göç kullanılarak ağ trafiği dengeleme algoritmasının ayrıntıları ve sistemin alt bileşenleri detaylı bir şekilde ele alınmıştır. Gerekli durumlarda gerçeklenme ayrıntılarına inilerek tez süresince gelişim gösteren kısımlar üzerinde durulmuştur. Ayrıca sistemin testinde kullanılan ve etmen tabanlı grid sistemini simüle eden yazılım bileşeni bu bölümde tanıtılmıştır. Bölüm sonunda etmen tabanlı grid üzerinde gerçekleştirilen testlerle sistemin çalışması gösterilmiş, sonuçlar elde edilmiştir ve elde edilen sonuçlar tartışılmıştır.

6. ve son bölümde önerilen sisteme ait çeşitli değerlendirmelere ve önerilere yer verilmiştir. Önerilen sistemin olası kullanım alanları tartışılmış, gelecekte yapılabilecek iyileştirmeler üzerine öneriler sunulmuştur.

# 2. ETMEN TABANLI GRİD SİSTEMİ

## 2.1 Bölümün Amacı

Bu bölümde etmen sistemleri ve grid sistemleri ile ilgili genel bilgiler verilmiştir. Kullanılan etmen tabanlı grid sisteminin mimarisi ve özellikleri anlatılmıştır. Tez kapsamında kullanılan etmen tabanlı grid sistemine yapılan eklentiler de bu bölümde ele alınmıştır.

## 2.2 Grid Sistemleri

Grid sistemleri temel olarak elektrik alanından gelen, elektrik dağıtımıyla ilgili bir kavramdır. Elektriksel grid sistemlerinde hizmet almak isteyen kullanıcılar dağıtılan enerjinin kaynağından, taşınma şeklinden ve dağıtım sisteminde yer alan herhangi bir karmaşık yapıdan bağımsız olarak sistemden faydalanırlar.

Yazılımsal grid sistemleri her çeşit bilgisayar kaynağının(işlem gücü, saklama alanı, özelleşmiş donanımlar) özelliklerinden bağımsız olarak hizmet almak isteyen kullanıcılara ortak bir ağ üzerinden paylaştırılmasını sağlar. Kullanıcılar kendi fiziksel konumlarından bağımsız olarak veya sistem altyapısı hakkında bilgi sahibi olmadan gridin sunduğu kaynaklardan hizmet alabilirler [6]. Bu bağlamda elektrik gridlerine benzetilebilirler. Yazılımsal gridler elinde bulundurduğu fiziksel olarak dağıtılmış ve heterojen bir yapıda olan tüm kaynakları sanal olarak tek bir kaynakmış gibi gösterir, kullanıcıların sunulan kaynak veya hizmetlere istenilen zamanda erişmesini sağlar [7].

Yazılım gridleri sundukları kaynakları ve hizmetleri düşük maliyetle ve yüksek performansla kullanıcılara ulaştırmayı hedeflemektedirler [8]. Yazılım gridleri büyük ölçekli heterojen kaynak paylaşımı ve yüksek performans hedefleri olan bir dağıtık sistem çeşididir.

İnternet üzerinde sanal organizasyonlar dağıtılmış konumlarda bulunan heterojen kaynakları, çeşitli hizmet ve verileri elektronik ticaret, bilimsel hesaplamalar, akademik çalışmalar gibi hedefleri gerçekleştirme doğrultusunda paylaşma ve

kullanma ihtiyacı duyarlar. İhtiyaç duyulan tüm bu servislerin tamamen farklı platformlarda yer alabileceği düşünüldüğünde böyle bir sistemde kaynakların, hizmetlerin ve veri akışının yönetilmesinin zorluğu görülmektedir [2].

Günümüzde uygulamalar kendilerine özgü kaynak kullanımı ve yönetim yapıları olan farklı teknolojilere ve farklı platformlara özgü bir şekilde tasarlanmış olabilmektedirler. Bu uygulamalar farklı platformlar ve/veya altyapılarda çalışması hedeflenecek şekilde çok çeşitli teknolojiler kullanılarak geliştirilmiş olabilirler. Ancak tüm bu farklılıklara rağmen bu uygulamaların uyum içinde, ortak bir amaca yönelik çalışmaları beklenmektedir. Heterojen kaynakların bu şekilde farklılaşmış uygulamalar arasında sorunsuz bir şekilde paylaşılmasını sağlamak grid sistemlerinin hedeflerindendir [2].

Yazılım gridleri önceleri şirketler, gruplar ve organizasyonlar bazında daha özelleşmiş bir yapıdayken, günümüzde farklı alanlarda uzmanlaşmış çeşitli organizasyonların birbirlerine sağladıkları uygulamalar, kaynaklar ve hizmetlerden gevşek bağlı bir yapıda fakat etkin bir şekilde faydalanma gereksinimi duymaları nedeniyle, organizasyonlar arası geniş çaplı, genel kullanımlı ve çok amaçlı bir hal almışlardır.

Grid sistemlerinin organizasyonların gevşek bağlı bir şekilde birbirleriyle entegre olabileceği geniş çaplı bir kullanım alanı sunmalarıyla birlikte bu altyapıyı sağlayabilmek adına bazı temel gereksinimleri ve özellikleri sağlamaları gerekli hale gelmiştir. Bu gereksinim ve özellikler şu şekilde sıralanabilir [6]:

- **Kaynakların etkin değerlendirilmesi:** Gridler boşta bekleyen kaynakların etkin olarak kullanılmasını sağlayarak kaynak kullanım oranını arttırırlar ve eldeki kaynakları etkin şekilde değerlendirirler. Bir uygulamanın grid üzerindeki işlem yükü düşük olan bir bilgisayarda çalıştırması buna örnek gösterilebilir.
- **Paralel CPU kapasitesi:** Yüksek CPU kapasitesi gerektiren bilimsel araştırmalara ek olarak bio-medikal hesaplamalar, finansal modellemeleri ve görüntü işleme gibi endüstriyel ve ticari çalışmalar yapılan alanlarda büyük CPU kaynaklarına ihtiyaç giderek artmıştır. Bu sebeple gridlerin sunduğu büyük CPU kapasitesi en çok ilgi çeken özelliklerinden olmuştur.
- **İşbirlikleri için sanal kaynak ve organizasyonların sağlanması:** Gridler

farklı amaçları olan katılımcılar arasında sanal kaynaklar ve organizasyonlar oluşturarak işbirliği ortamını temin ederler.

- **Ekstra kaynaklara erişim:** CPU ve disk kapasitesi gibi öncül kaynakların yanı sıra bilgisayar sistemlerine ait bant genişliği veya özelleşmiş donanımlar gibi diğer kaynaklara da grid tarafından erişim sağlanabilir.
- **Yük Dengeleme:** Gridler etkili yük dengeleme algoritmaları kullanarak iş atamalarının sistem kaynakları kullanımını maksimize edecek şekilde yapılmasını sağlarlar.
- **Güvenilirlik:** Geleneksel yüksek kapasiteli hesaplama sistemleri güvenirliliği sağlamak adına pahalı donanımlar kullanmaktadırlar. Sistemdeki depolama aygıtları, güç üniteleri ve işlemciler gibi hayati bileşenlerin yedeği alınarak, pahalı bir yol da olsa, oluşacak hataların en az kayıpla atlatılması sağlanmaktadır. Gridler fiziksel olarak dağıtılmış katılımcılara sahip sistemler olduğundan sistemin herhangi bir yerinde meydana gelebilecek herhangi bir arıza sistemin geri kalanını etkilememektedir. Grid yönetim yazılımları bir arıza durumunda yarım kalan işleri otomatik olarak arızadan etkilenmeyen bir katılımcıya atama yeteneğine sahiptirler.
- **Yönetim:** Grid yönetim yazılımları sistemdeki kaynak durumlarının ve yük dağılımlarının izlenebildiği ara yüzler sunmaktadırlar.

## 2.3 Etmen Sistemleri

Günümüzde tek bilgisayarda çalışan statik programlardansa değişik konumlarda bulunan farklı altyapılara sahip dağıtılmış ortamlarda çalışabilen esnek yazılımlar daha revaçtadırlar. Etmen sistemleri bilgisayar ağları, web servisleri, yazılım mühendisliği, nesneye yönelik programlama, yapay zekâ, dağınık ve paralel sistemler, mobil sistemler, kontrol ve karar verme sistemleri, bilgi çıkarımı ve elektronik ticaret uygulamaları gibi birçok bilişim alanında kullanılmaktadır [9].

### 2.3.1 Etmenler

Etmenler kendilerine atanmış görevleri yerine getirmek için belirli ortamlara yerleştirilmiş, çevreleriyle esnek ve otonom ilişkiler kurabilen bilgisayar sistemleridir [1,10]. Etmenler çevresiyle sürekli etkileşim içinde olabilen bu

etkileşimler sonucunda çıkarsamalarda bulunabilen, öğrenebilen ve işbirlikleri kurarak ortaklaşa hareket edebilen yazılım bileşenleri olarak dikkat çekmektedirler [9,10].Etmen sistemlerine ait temel özellikler şu şekilde sıralanabilir. [1,10]

- **Otonomluk:** Etmenler çevreleriyle doğrudan etkileşime geçmeden elinde bulundurduğu kaynaklar, tuttukları değerler ve yaptıkları hesaplamalarla kararlar verebilirler.
- **Duyarlılık:** Etmenler çevrelerini algılama ve meydana gelen değişikliklere zamanında tepki vererek çevrelerinde değişiklikler yapabilme yeteneğine sahiptirler.
- **Akıllı davranışlılık:** Etmenler çevreleriyle sadece basit davranışlarla etkileşimde bulunmazlar. Belirli bir amaca yönelik olarak bir stratejik plan çerçevesinde hareket ederler.
- **Sosyallik:** Etmenler diğer etmenlerle ve bazen de farklı arayüzler vasıtasıyla insanlarla çeşitli standartlardaki iletişim biçimleriyle iletişime geçebilirler, amaçlarını yerine getirmeye yönelik işbirlikleri kurabilirler.

### 2.3.2 Çoklu Etmen Sistemleri

Etmenler çevrelerini algılama ve çevrelerinde algıladıkları olaylara tepki vermenin yanında çevreleriyle sosyal ilişkiler kurabilen aktif ve yaşayan yazılım bileşenleridir. Etmenler ortaklık, yardımlaşma ve rekabet ilişkileri veya bu ilişki şekillerinin birleşimi şeklinde ilişkiler kurabilirler. Çoklu etmen sistemleri birden fazla etmenin birbiriyle iletişim kurmalarıyla oluşturdukları sistemlerdir. Etmenler bu şekilde bir araya gelerek oluşturdukları sistemlerle tekil olarak çözemeyecekleri problemleri çözebilirler. Çoklu etmen sistemlerinin temel özellikleri şu şekilde sıralanabilir [1,11].

- Etmenlerin problemin çözümü hakkında eksik bilgileri veya yetenekleri vardır.
- Tüm sistemi kontrol edemezler.
- Veri merkezi değildir, bir noktadan problemin çözümü için gerekli olan tüm veriye ulaşmak mümkün değildir.
- Hesaplamalar asenkron olarak gerçekleştirilir.

### 2.3.3 Etmenler Arası İletişim

Etmen sistemleri etmen iletişim mekanizmalarını sağlayabildikleri ölçüde güçlü ve etkin kullanılabilir hale gelirler. Sosyal yapıları gereği etmenler diğer etmenlerle, çeşitli kaynaklarla, birtakım arayüzler vasıtasıyla insanlarla yani genel olarak çevreleriyle sürekli iletişim içinde olmaları gerekmektedir. Kullanım alanları yaygınlaştıkça farklı tiplerdeki etmenlerin iletişimini sağlayabilmek için standartlaşmış ve gelişmiş iletişim yöntemlerine ve dillerine ihtiyaç duyulmuştur.

Günümüzde yaygın olarak kullanılan iki etmen iletişim dili ön plana çıkmaktadır, KQML ve FIPA ACL. KQML (Knowledge Query and Manipulation Language) ARPA Bilgi paylaşım grubu tarafından geliştirilmiştir, etmenler arası bilgi ve mesaj aktarımında kullanılan bir dil ve protokol tanımlar [1,12]. FIPA ACL, FIPA tarafından geliştirilen açık standartları tanımlanmış bir mesajlaşma dilidir [13].

## 2.4 JADE Ana Çatısı

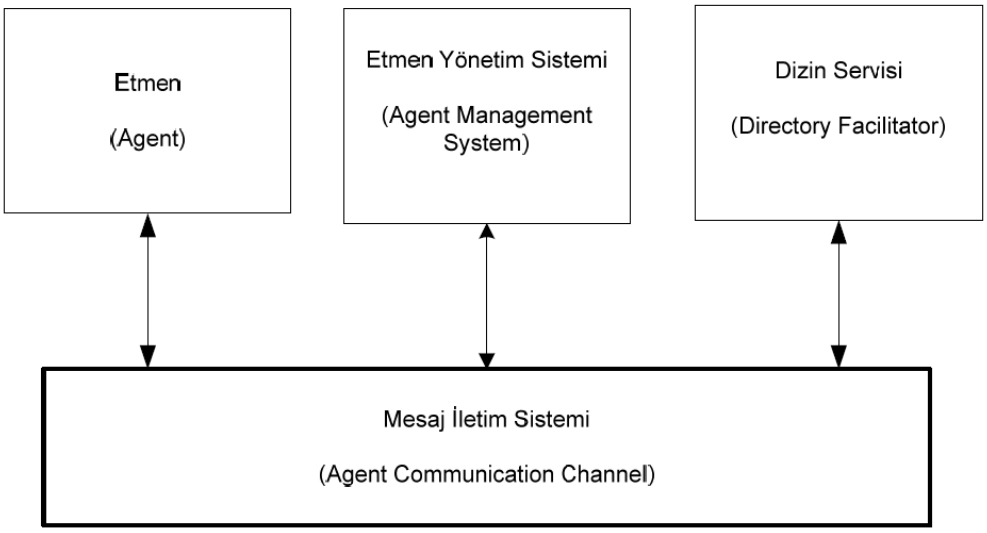
JADE - Java Agent Development Framework (Java Etmen Geliştirme Ana çatısı) FIPA standartlarına tam uyumlu, çoklu etmen sistemi geliştirme ana çatısıdır. JADE biri FIPA uyumlu etmen platformu, diğeri ise Java ile etmen geliştirme paketi olmak üzere iki ana yapıdan oluşur. JADE, tamamıyla Java dilinde yazılarak bu dilin esnek yapısını ve soyut ara yüzlerini, hazır paketlerini kullanarak kolaylıkla etmen geliştirme yapısı sunmayı hedeflemiştir [1,14]. JADE kullanıcılarına ve geliştiricilerine FIPA standartlarının tanımladığı ve gerektirdiği özellikleri, platformları ve yönetim katmanlarını hazır olarak sunmaktadır [14].

FIPA tarafından önerilen standart bir etmen sisteminin bileşenleri Şekil 2.1'de görülebilinir [1,14].

AMS – Agent Management System – Etmen Yönetim Sistemi denilen bir etmen tüm etmen platformunun yönetiminden sorumludur. Her sistemde bir ve yalnız bir tane AMS bulunur. AMS etmenler için yaşam döngüsü servisleri sunar, etmenlere kimlik atanması (AID) ve durumlarının takip işlemlerini yapar. Platforma bağlanacak her etmen AMS den geçerli bir kimlik almak zorundadır.

DF (Directory Facilitator – Dizin Servisi) Etmenler arası sarı sayfalar servisi sunar

Mesaj İletim Sistemi – ACC (Agent Communication Channel) platform içinde mesaj iletimi yapısını kuran bileşendir.




**Şekil 2.1 :** FIPA standartlarında etmen sistemi.

JADE FIPA standartlarıyla ve yukarıda tanıtılan mimariyle tam uyumludur. JADE platformu çalıştırıldığında AMS, DF ve ACC bileşenleri yaratılır ve çalıştırılır. Aynı zamanda bu bileşenlerin her biri farklı bilgisayarlarda çalıştırılabilecek yapıya sahiptirler [1,14].

## 2.5 Etmen Tabanlı Grid Sistemi

Grid sistemleri genel olarak dağıtık yapının güvenliğine, güvenilirliğine ve bileşenlerinin uyum içinde birlikte çalışabileceği bir alt yapıya önem verirler. Gridlerin ana amacı dinamik ve çok katılımcılı sanal organizasyonlarda kaynak paylaşımı ve problem çözme sistemleri yaratmaktır [3,1]. Bu nedenden ötürü grid sistemlerinin tanımladığı protokollerde veri güvenliği, kararlılık daha çok öne çıkan tasarım isterleri olarak öne çıkmaktadır.

Etmenler ise daha kaygan zeminler üzerinde inşa edilmiş sistemleri oluşturmaktadırlar. Etmen sistemleri esnek ve bağımsız çalışan yazılım bileşenleri oluşturmaya daha çok yoğunlaşmışlardır. Dolayısıyla etmenler sistem ve problem hakkında sahip oldukları sınırlı bilgi ve birikimle bireysel işlemlerini yaparken tüm sistemin kazanımın yanında kendi kazanımlarını da düşünen sosyal yazılım bileşenleridir [3,1]. Etmenlerin tasarımında esas tasarım kriteri esnekliktir. Klasik etmen sistemlerinde katılımcı etmenler dağıtık yerlerde yaptıkları işler sistemin ana amacı için önemli olsa da varlığı tüm sistemin oluşturulma amacı için kesin gerekli değildir.

Etmen ve grid sistemleri sanal organizasyonlar veya işbirlikleri kuran dağıtık

sistemlerin tasarım amaçlarına göre özelleşmiş farklı şekilleri olarak ele alınabilir. Tezimizde kullandığımız etmen tabanlı grid sistemi, etmen ve grid sistemlerinin karakteristik özelliklerini birleştirerek esnek, mobil, kararlı, güvenli bir dağıtık sistem olarak karşımıza çıkmaktadır [1].

### 2.5.1 Sistemin Temel Yapısı

Kullanılan etmen tabanlı grid sistemi JADE ana çatısı kullanılarak gerçeklenmiştir [1]. Sistemin JADE kullanılarak gerçeklenmesi sayesinde grid sisteminin FIPA standartlarına uyumlu halde yazılmış başka etmenlerle kolaylıkla entegre olabilmektedir. FIPA ACL(etmen iletişim dili) kullanılarak etmenler arası iletişim kurulabilmektedir. Ayrıca JADE'in sağladığı hazır uygulamalar hata ayıklama ve geliştirme işlemleri sırasında kolaylıklar sağlamaktadır.

Sistemde yer alan temel yazılım paketleri ve içerikleri şunlardır:

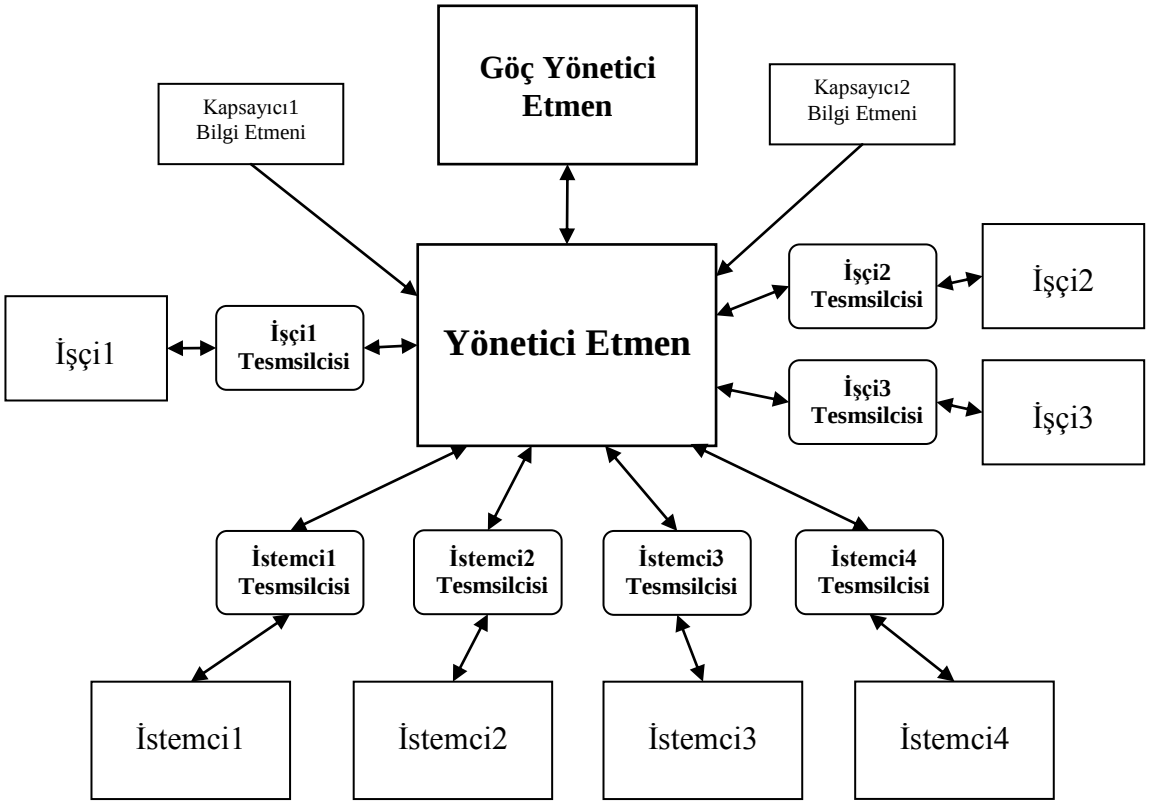
- grid: Grid sisteminin yönetimsel sınıflarını içeren pakettir.
- client: İstemci etmenlere ilişkin sınıfları tutar.
- worker: İşçi etmenlere ilişkin sınıfları tutar.
- delegate: Temsilci etmenlere ilişkin sınıfların bulunduğu pakettir.
- message: Etmenler arası mesajlaşma işlemlerine ilişkin sınıfları içerir.
- task: İstemciler tarafından sisteme gönderilen ve işçi etmenlerce yürütülen görevlere ilişkin sınıflar bu pakettedir.
- property: Görevlerin ihtiyaç duydukları ve işçi etmenlerin sahip oldukları özellikleri gerçekleyen sınıfları barındırır.
- pooler: Delegelerin ilgili etmenin halen sisteme bağlı olup olmadığını test etmeleri gerekmektedir. Bu işlem için gereken protokol bu paketteki sınıflarca yürütülür.
- test: Test uygulamaları bu pakettedir.

Bunlara ek olarak tezimizde gride eklemiş olduğumuz paketler şunlardır:

- migration: Görevler arası mesajlaşmaları izleyen ve görev göçü için karar algoritmasını gerçekleyen sınıfları barındırır.
- testMigrate: Görev göçü test uygulamaları gerçekleştiren sınıfları içerir.
- container: Kapsayıcılarla ilgili sınıfları içerir

Etmen tabanlı grid sisteminde, sisteme bağlanan istemcilerin bölümlendirip, sisteme verdikleri işlerin uygun biçimde işletilip kendilerine bildirilmesi, etmenlerin yeterli

ve gerekli esneklikle sisteme bağlanması ve sistemden ayrılması, hata durumlarının tespit edilip giderilmesi işlemlerine ilişkin yöntemler ve protokoller tanımlanmıştır. Bu yapıya ek olarak tezimizde önerdiğimiz görev göçü mekanizmasının gerçeklenmesi için gerekli altyapı ve protokoller de sisteme entegre edilmiştir. Sistemin alt bileşenleri izleyen bölümlerde anlatılmıştır. Şekil 2.2 :'de sistemin genel mimarisi gösterilmiştir.




**Şekil 2.2 :** Sistemin genel mimarisi.

### 2.5.2 Yönetici bileşenleri

Kullanılan etmen tabanlı grid sisteminde tüm akışın kontrolünü yapan bir yönetici etmen bulunmaktadır. Sistemde yanlızca bir tane yönetici etmen bulunmaktadır Sunucu makinede grid yaratılırken yönetici etmen de yaratılır. Yönetici etmenin görevleri şu şekilde sıralanabilir;

- Sisteme bağlanan istemci ve işçi etmenlerin kayıtlarını tutmak ve onlar bağlandığında her biri için bir temsilci etmen yaratmak.
- Etmenlerin sistemden güvenli biçimde ayrılmalarını sağlamak.
- İşçi etmenlerin üzerlerindeki işleri tamamlamadan sistemden ayrılmaları durumunda ilgili görevin tekrar atanmasını sağlamak

- İstemci etmenlerin sistemden ayrılarak görevlerinin sonuçları istemedikleri durumunda ilgili işçi etmenlerin çalışmalarını sonlandırmak
- Tezimiz kapsamında sisteme eklediğimiz görev göçü izleyici-yönetici etmeni yaratarak sisteme dahil etmek.

Yönetici etmen bağlanan her bir etmen için (işçi/istemci) bir temsilci (delegate) etmen yaratır. Temsilci etmenler temsil ettikleri işçi ya da istemci etmenle olan tüm iletişimi ve protokol gerçeklemelerini yaparlar.

Tezimiz kapsamında yönetici etmene göç işlemini gerçekleştirmesini sağlayacağı özellikler kazandırılmıştır. Bu kapsamda yönetici etmen yaratıldığında görev göçü izleyici-yönetici etmeni(göç yönetici etmen) yaratarak sisteme dahil eder. Bunun yanında yönetici etmene yaptığımız eklemeler sayesinde göç hesaplaması sonucu edinilen talimatlara göre grid içerisindeki işçi etmenler tarafından çalıştırılan görevlerin nereden nereye göç edeceği bilgisini ilgili etmenlere iletir ve göç işlemini koordine eder.

### 2.5.3 Sistemdeki Etmenler

Tezimizde kullandığımız grid sisteminde yönetici etmene ek olarak istemci, işçi , istemci temsilcisi, işçi temsilcisi etmenleri ve tezimiz kapsamında sisteme eklediğimiz göç yönetici etmen bulunmaktadır. Görev göçünü sağlamak üzere sistemde hazır bulunan etmenler üzerinde de bazı değişiklikler yapılmıştır. Aşağıda sistemde yer alan bu etmenlerin görevleri ve yapılan değişikler aşağıda anlatılmıştır.

**İstemci Etmenler:** İstemciler sistemden hizmet alan etmenlerdir. İstemci etmenler sistemin kendilerini tanıması için gereken bilgileri yönetici etmene yollayarak gride bağlandıktan sonra yürütmek istedikleri görevleri bildirirler. Grid sistemi kendi yaşam döngüsü içinde bu görevleri yürütüp sonuçlarını ilgili istemcilere bildirir.

Tezimiz kapsamında istemci etmenlerde yapısal bir değişiklik yapılmamıştır, etmen tabanlı grid sistemi tarafından tasarlandığı şekliyle kullanılmıştır.

**İstemci Temsilcileri:** Her istemci için yönetici etmen tarafından bir temsilci etmen yaratılır. Bu temsilci etmen temsil ettiği istemci ile grid yönetimi arasındaki iletişimi koordine eder. İstemci temsilci etmenlerinde tezimiz kapsamında değişiklik yapılmamıştır.

**İşçi Etmenler:** İşçi etmenler gönüllü olarak grid sistemine bağlanan ve sahip

oldukları kaynakları gridin kullanımına sunan etmenlerdir. Önerilen sisteme yönelik yapılan çalışmada işçi etmenler işlemci gücüne dair kaynakları paylaşmaktadır. İşlemci gücünün yanında diğer hesaplama birimlerinin (grafik kartı v.b.) kaynakları da işçiler tarafından paylaşılabilinir. Ancak oluşturulan işçilerin depolama alanı gibi kaynaklarını paylaşımı çalışmanın kapsamının dışında tutulmuştur. İşçi etmenler sahip oldukları donanım-yazılım özelliklerine ve yeteneklerine uygun olarak kendilerine atanan görevleri işletip sonuçlarını ilgili etmenlere bildirirler.

Tezimizde işçi etmenler etmen tabanlı grid sistemindeki tasarlandığı yapısına ek olarak, görev göçünü sağlayabilecek şekilde özelleştirilmiştir. İşçi etmenlere çalıştırdıkları görevleri başka bir etmene aktarabilecek, kendisine başka işçi etmenler tarafından aktarılmış olan görevleri devam ettirebilecek eklemeler yapılmıştır.

Tezimiz işçi etmenler tarafından çalıştırılan görevler arası mesaj alışverişinin analizinin sonuçlarına göre görev göçü sağlanarak ağ yükünü dengelemeyi amaçlamaktadır. Bu amaca yönelik olarak, işçiler tarafından işletilen görevlerin birbirleriyle mesajlaşabilme mekanizmaları işçi etmenlere eklenmiştir.

**İşçi Temsilcileri:** İstemci ve temsilcileri arasındaki ilişkiye benzer bir ilişki işçi etmenleri ve işçi temsilci etmenleri arasında da bir ilişki vardır. İşçi etmenleri grid yönetimiyle olan tüm iletişimi kendisi için yaratılıp sisteme dahil edilen temsilci etmenler üzerinden yapar.

Tezimiz kapsamında görev göçünü sağlamak adına işçi temsilcisi etmenlere bazı eklemeler yapılmıştır.

**Göç Yönetici Etmen:** Göç yönetici etmen tezimiz kapsamında sisteme eklenmiştir. Bu etmen görevler arasında yapılan mesajlaşmaları takip eder ve belirli aralıklarla bu mesajlaşma trafiğini analiz ederek etmenlerde çalışan görevlerin hangi etmenlerde çalışırlarsa ağ trafiğinde bir iyileşme sağlanabileceğini hesaplar. Hesaplamalar sonucunda aldığı göç kararlarını ilgili görevlerin göç işlemlerini gerçekleştirmesi için yönetici etmene bildirir. Göç yönetici etmenin yapısı ve diğer etmenlerle ilişkisi

bölüm 2.5.4 ‘de detaylı olarak ele alınacaktır.

**Kapsayıcı Bilgi Etmeni:** Jade etmenleri kapsayıcı (jade.wrapper.AgentContainer) olarak tanımlayabileceğimiz yapılara eklenerek grid sistemine dahil olurlar. Tezimizde işçi etmenlerin bağlandıkları kapsayıcıların çalıştıkları makinede sisteme sunulan kaynakları yönetici etmene bildirmeleri gereklidir. Bu bildirimin yapılması

için işçi etmenlerin bağlanacağı kapsayıcılar yaratıldığında bu bilgiyi gönderecek bir kapsayıcı bilgi etmeni yaratılıp kapsayıcıya bağlanarak sisteme dahil edilir. Bu etmen çalışmaya başladığında yönetici etmene ilgili makinede sunulan kaynakların bildirimini yapar. Grid işçilerin dağıtımını ve göç kararlarını gerçekleştirirken bu bilgilerden faydalanır.

### 2.5.4 Göç Yönetici Etmen

Göç yönetici etmen tezimizin ana bir unsuru olan ağ trafik yükünü dengeleme işlemini gerçekleştirdiğinden daha detaylı olarak ele alacağız. Göç yönetici etmen sistemi izleyerek ağ trafik yükü dengeleme algoritmasını işleten etmendir. Göç yönetici etmen grid.Agent sınıfından türetilmiş bir etmen sınıfıdır.

Göç yönetici etmen içerisinde görevlere ait tanımlayıcıları ve bu tanımlayıcılarla eşleştirilmiş mesajlaşma bilgilerini tutar. Her bir görev tarafından gönderilen mesajlaşma bilgisini o göreve ait tanımlayıcı ile eşleyip ağ trafiği dengeleme algoritmasında kullanılmak üzere bir haritaya kaydeder. Farklı görevler tarafından gönderilen mesaj bilgilerinin alınıp kaydedilmesi sırasında haritaya erişim karşılıklı dışlamalı olarak gerçeklenmiştir. Ayrıca bir görevin göç ettirilmesinin da sisteme getireceği tahmini taşıma maaliyetleri de göç yönetici etmen içerisinde tanımlıdır. Bu maaliyet tezimiz kapsamında sabit ve tüm görevlerin herhangi bir kapsayıcıdan diğerine göç etmesi için aynıdır.

Göç yönetici etmen yaratılırken migration.MigrationAnalyser tipinden bir nesne yaratılarak Jade'in jade.core.Agent sınıfında sunduğu *addBehaviour* yordamı kullanılarak göç yönetici etmene özellik olarak eklenir. *addBehaviour* yordamı jade.core.behaviours.Behaviour tipinden nesneleri kabul eder. Herbir Behaviour arayüzüne sahip nesne, etmenler için belli işlemleri yerine getiren birer alt davranış olarak düşünülebilir. Jade bazı standart etmen davranışlarına ait Behaviour sınıfları gerçeklemiştir. MigrationAnalyser nesnesi Behaviour sınıfından türemiş jade.core.behaviours.TickerBehaviour sınıfından türetilmiştir. TickerBehaviour sınıfı belirlenen zaman aralıklarında *onTick* yordamına çağrı yapma davranışını gerçeklemiş bir sınıftır.

Göç yönetici etmenin çalışması şu döngüyü tekrarlar:

1. İşçi etmenler tarafından belirli aralıklarla gönderilen, görevlerin birbirleriyle yapmış olduğu mesajlaşma bilgilerini içeren mesajları alarak daha önce

belirtilen haritalara kaydeder. Mesajlaşmaların bildirilmesi ile ilgili protokol 5.1.4.5'te ayrıntılı olarak ele alınmıştır.

2. Biriken mesajlaşma bilgilerini kullanarak ağ trafiği dengeleme algoritmasını çalıştırır. Bu işlem MigrationAnalyser sınıfı tarafında gerçekleştirilir. MigrationAnalyser sınıfı *onTick* yordamında göç yönetici etmen içerisindeki *analyseMigrations* yordamına çağrı yapar. Bu sayede göç yöneticinin belirli aralıklarla ağ trafiği dengeleme algoritmasını çalıştırılması sağlanmış olur. Ağ trafiği dengeleme algoritmasının çalıştırılması ile ilgili protokol 5.1.4.6'da ayrıntılı olarak ele alınmıştır.
3. Ağ trafiği dengeleme algoritması sonucunda gerekli görülen görev göçleri yönetici etmene bildirilir. Yönetici etmen ile göç yönetici etmen birbirlerine ait referansları bulundurduklarından bu sayede yazılımsal olarak birbirlerine doğrudan erişebilir olduklarından gerekli görülen göç bildirimleri mesajlaşma ile değil göç yönetici tarafından yönetici etmen yapılan doğrudan yordam çağrısı şeklinde gerçekleştirilir.

Göç yönetici etmen yaşam döngüsü boyunca sistemi analiz ederek çeşitli göç kararları verir ve bu şekilde ağ üzerindeki yükü dengeler.

### 2.5.5 Etmenler Arası Mesajlaşma

Mesajlaşma dağıtık sistemlerin en önemli noktalarındandır. Etmen tabanlı grid sisteminde güçlü bir mesajlaşma sistemi geliştirilmiştir [1]. Mesajlaşma yapısında JADE ACL modülü kullanılmıştır [1,14]. Etmen tabanlı grid sistemin sistemin temel işlemlerini sağlayacak mesaj tipleri tanımlanmıştır. Sistemde tanımlanmış olan sisteme bağlanma (CLIENT_REGISTER, WORKER_REGISTER), sisteme kabul edilme (CLIENT_ACCEPTED, WORKER_ACCEPTED), sistemden ayrılma (CLIENT_UNREGISTER, CLIENT_DONE, WORKER_UNREGISTER), bilgi gönderme (CLIENT_INFO, WORKER_INFO), yeniden bağlanma (CLIENT_RECONNECT), sonlandırılma (STOP_WORKER, WORKER_STOPPED), uygunluk bildirimi (WORKER_FREE), görev isteği (TASK_REQUEST), görev kabulü (TASK_REQUEST_ACCEPTED), görev iptali (TASK_REQUEST_FAILED), görev tamamlanması (TASK_COMPLETED) gibi sistemin çalışması ve sürdürülmesi ile ilgili temel mesajlar tezimiz kapsamında da

kullanılmıştır [1]. Bu mesajlara ek olarak görev göçünü sağlamak amacıyla tanımlanan mesajlar aşağıda anlatılmıştır:

- TASK_MIGRATE_REQUEST: Mesaj içeriğine eklenmiş olan görevin göç ettirilmesi isteğini bildirir.
- TASK_MIGRATED: Görevin göç işleminin tamamlandığını iççi temsilcilerine ve yönetici etmene bildirir.
- TASK_MESSAGE: İşçilerin çalıştırdıkları görevlerin kendi aralarında mesajlaşmalarını sağlar.
- TASK_RECEIVERS_LIST_REQUEST: Çalıştırılan görevin mesaj gönderdiği görevlerin hangi işçiler tarafından çalıştırıldığı bilgisini talep eder.
- TASK_SENDERS_LIST_REQUEST: Çalıştırılan görevin mesaj aldığı görevlerin hangi işçiler tarafından çalıştırıldığı bilgisini talep eder.
- TASK_RECEIVER_AGENTLIST: Çalıştırılan görevin mesaj gönderdiği görevlerin hangi işçiler tarafından çalıştırıldığı bilgisini içerir.
- TASK_SENDER_AGENTLIST: Çalıştırılan görevin mesaj aldığı görevlerin hangi işçiler tarafından çalıştırıldığı bilgisini içerir.
- TASK_CONNECTED: Görevin işçi etmen tarafından çalıştırılmaya başlandığını bu göreve mesaj gönderecek görevlerin çalıştırıldıkları etmenlere bildirir.
- TASK_MIGRATE_ANALYSE_MESSAGE: Bir işçinin çalıştırdığı gçrevin diğer etmenlerle mesajlaşma bilgisini göç yönetici etmene gönderir.
- TASK_MIGRATION_ACCEPTED: İşçi etmenler kendilerine göç ettirilen görevin çalıştırılmaya devam ettirileceği bilgisini yönetici etmene iletir.
- TASK_MIGRATION_INFORMATION: Göç gerçekleştiren grevin mesajlaştığı görevlere göç durumunu bildirir.
- MIGRATION_MANAGER: İşçi etmen sisteme dahil olduğunda göç yönetici etmene ait bilgileri işçi etmene iletir.
- CONTAINER_INFO: Kapsayıcı yaratıldığında yönetici etmene kapsayıcı ile ilgili bilgileri gönderir.

Etmen tabanlı grid sistemi etmenlerin sisteme çok fazla yük olmadan kendilerine

gönderilen mesajı kolayca ve hızlıca algılayıp gerekeni yapabileceği bir altyapı sağlamaktadır. Bu mesajlaşma altyapısı JADE tarafından sınıflar kullanılarak gerçeklenmiştir. Sistem tarafından sunulan mesajlaşma altyapısı etmenlere meşgul durumda bekletmeden (busy waiting) kendilerine iletilen mesajı seçerek alması ve işlemesi sağlamaktadır [1].

Etmen tabanlı grid sisteminin mesajlaşma altyapısı tarafından kullanılan mesajlar JADE mimarisinin sunduğu jade.lang.acl.ACLMessage sınıfından türetilmiştir. Bu sınıfın FIPA standartlarına uygun bazı nitelikleri vardır. Etmenler aşağıda tanımları verilen bu nitelikleri göndermek istedikleri mesaj türüne göre belirleyip göndermektedirler [1].

- Edimsel (Performative): Mesajın türüne dair bilgi içerir. Yukarıda tanıtılan mesaj tiplerinin her biri kendisi için bir edimsel tanımlamışlardır.
- Ontoloji (Ontology): Mesajın içeriğine dair bilgi içerir. Sistemde kullanılan mesaj tiplerinin her biri kendileri için bir ontoloji tanımlamışlardır.
- İçerik (Content): Mesajın içeriğidir. FIPA standartlarına göre mesajların içerikleri metin tabanlıdır. Ancak JADE Java dilinin bir yapısı olan serileştirilebilir (Serializable) nesneleri de mesaj içeriği olarak kullanabilmektedir.

Tezimiz kapsamında görev göçünün gerçekleştirilmesinde serileştirilebilir Java nesnelerinin mesaj içeriği olarak kullanılabilmesinin önemli bir yeri vardır. Görevler bir işçi etmenden diğerine serileştirilerek gönderilmektedir. Göç eden görevlerin bağlam bilgileriyle birlikte iletilmesi bu sayede sağlanmaktadır. Bağlam bilgisinin mesaj içeriğine eklenebilecek serileştirilmiş bir formata dönüştürülmesi işlemi bölüm 3'te ele alınmıştır.

### 2.5.6 Görev İşletimi

Grid sisteminde görevler ve özelliklerinin esnek biçimde gerçeklenmesi gerekmektedir. Kullanılan etmen tabanlı grid sisteminde görevlerin gerçeklemeye ve uygulamaya göre özelleştirilebilmektedir. Bazı görevlerin yerine getirilmesi için özel donanımlar (ekran kartı, belirli bir giriş/çıkış kartı) ya da hazır yazılımların bulunması gerekebilir. Yönetici etmen görevi yönetecek işçi etmeni seçerken bu ekstra özelliklere de dikkat ederek atamaları yapmaktadır.

Etmen tabanlı grid sisteminde çalıştırılmak istenen görevlerin soyut task. Task sınıfıyla türetilmesi gerekmektedir. task. Task sınıfından türetilen sınıflar soyut olan

execute yordamını yapmak istedikleri işe göre gerçeklerler. Bu sayede her amaca yönelik görev tanımlamak mümkün olabilmektedir. Tezimizde bu yapı örnek alınmıştır ancak tezimizin gerekesinimleri gözönünde bulundurularak değiştirilip yeniden gerçeklenmiştir. Tezimiz kapsamında kullandığımız bağlam bilgisinin taşınabilmesi için task.Task sınıfının java.lang.Runnable ve java.io.Serializable arayüzlerini gerçeklemeleri gerekmektedir. java.lang.Runnable arayüzünün gerçeklenmesi ile görevlerin yaptıkları işler java.lang.Runnable arayüzünün run yordamı altında yapılmaya başlanmıştır. Bu sebeple tezimizde grid sisteminde görevlerin tanımlandığı execute yordamının yerini run yordamı almıştır. Bunun görev göçünün sağlanmasına bağlı sebepleri 3.bölümde ve 5.bölümde daha detaylı olarak ele alınacaktır.

Görevlerin gerek duydukları özellikler de property.Property ve property.PropertyList sınıflılarıyla gerçeklenmiştir. Görevler yürütülmek için ihtiyaçları olan özellikleri (Property) bir liste halinde (PropertyList) tutarlar. İşçi etmenlerde sahip oldukları özellikleri aynı yolla tutarlar. Bir görev bu şekilde belirli özelliklere ihtiyaç duyuyorsa soyut olan properties yordamını gerçekleyerek bu özellikleri belirtir. Yönetici etmen görev atamasında işçi ve görevin özelliklerinin uyumlu olmasına dikkat eder. İşçi etmen bir görevi aldığında bu görevin execute yordamını çağırarak görevin yerine getirilmesini sağlar.

Tezimiz kapsamında görevlerin birbirleriyle mesajlaşmalarını ve göç mekanizmasını sağlamak için task.Task soyut sınıfına birtakım eklemeler yapılmıştır. Görevler mesajlaştıkları görevlerin hangi işçi etmenler tarafından çalıştırıldıkları bilgilerine sahip hale getirilmişlerdir. Bu sayede görevler mesajlaşırken hangi etmenlere mesaj göndereceklerini bilmektedirler. Bir görevin ilk kez çalıştırıldığı zaman baştan başlatılması, göç ettirildiğinde ise kaldığı yerden devam ettirilmesi mekanizması alt seviyede task.Task sınıfına eklenmiştir. Görevi çalıştıran veya devam ettiren etmen bu durumdan bağımsız olarak task.Task sınıfına eklenmiş olan startRunning yordamını çağırmaktadır. Böylelikle işçi etmenler görevin baştan başlatılması veya kaldığı yerden devam ettirilmesi işlemlerinin nasıl gerçeklendiği ile ilgilenmemektedir.

### 2.5.7 Protokoller

Etmen tabanlı grid sisteminde gridin yaşam döngüsü ve işleyişiyle ilgili temel protokoller tanımlanmıştır [1]. Etmenlerin sisteme bağlanma protokolleri, görev

atama protokolü, etmenlerin sistemden ayrılma protokolleri değiştirilmeden kullanılmıştır [1]. Tezimiz kapsamında etmen tabanlı grid sisteminin bazı tanımlı protokollerini kullanılmakla birlikte mesajlaşma analizi ve görev göçü gibi işlemleri sağlamak için sistemde yeni protokoller tanımlanmıştır. Görevlerin yürütülmesi protokolüne değişiklikler yapılarak görevler arası mesajlaşmaların sağlanması ile ilgili eklemeler yapılmıştır. Yeni tanımlanan ve değiştirilen protokoller bölüm 5'de detaylı olarak ele alınacaktır.

Etmen tabanlı grid sisteminde tanımlanmış ve tez kapsamında değiştirmeden kullandığımız bazı protokoller bölümün kalan kısmında anlatılmaktadır.

#### 2.5.7.1 Etmenlerin sisteme bağlanma protokolü

Etmen tabanlı grid sisteminde etmenler JADE ana çatısına bağlandıklarında kendilerine bir kimlik numarası (AID) tahsis edilir ve böylece sistemde çalışmaya başlarlar. Ancak etmenlerin JADE sistemine dâhil olması önerdiğimiz grid sistemi için yeterli değildir. Yönetici etmenin bağlanan etmenin türü ve özelikleri hakkında bilgi sahibi olması gerekmektedir. Aynı zamanda yönetici etmen bağlanan etmenin gride dâhil edilip edilmeyeceğine de karar vermelidir (örneğin belirli güvenlik ve kimlik doğrulama yöntemleriyle). Böyle bir karar yapısı bu tezin konusu dışında kalmakla beraber sistemin ileride böyle desteklere sahip olabilmesi için protokol esnek tasarlanmıştır [1].

Sisteme bağlanabilecek iki tür etmen vardır; istemci ve işçi etmenler. Yönetici etmen grid sistemi oluşturulduğunda sistemin bir parçası olarak oluşturulur. Temsilci etmenler ise sisteme bağlanan her bir işçi ya da istemci etmen için yönetici etmen tarafından yaratılarak sisteme dâhil edilirler ve dolayısıyla söz konusu bağlanma protokolüne tabii değildirler [1].

**İstemci etmenin sisteme bağlanma protokolü**

İstemci etmen sisteme bağlanıp sistemden çıkabilir ve bir süre sonra tekrar bağlanma talebi gönderebilir. Böylece istemci etmenin yürütülmek üzere sisteme görevleri göndermesi, tüm görevleri gönderdikten sonra sistemden ayrılması ve bir süre sonra sisteme yeniden bağlanarak tamamlanan görevlerin sonuçlarını alabilmesi sağlanmaktadır [1].

Bağlanma protokolün işleyişi şu şekildedir.

1.İstemci etmen sisteme bağlanma isteğini yönetici etmene CLIENT_REGISTER

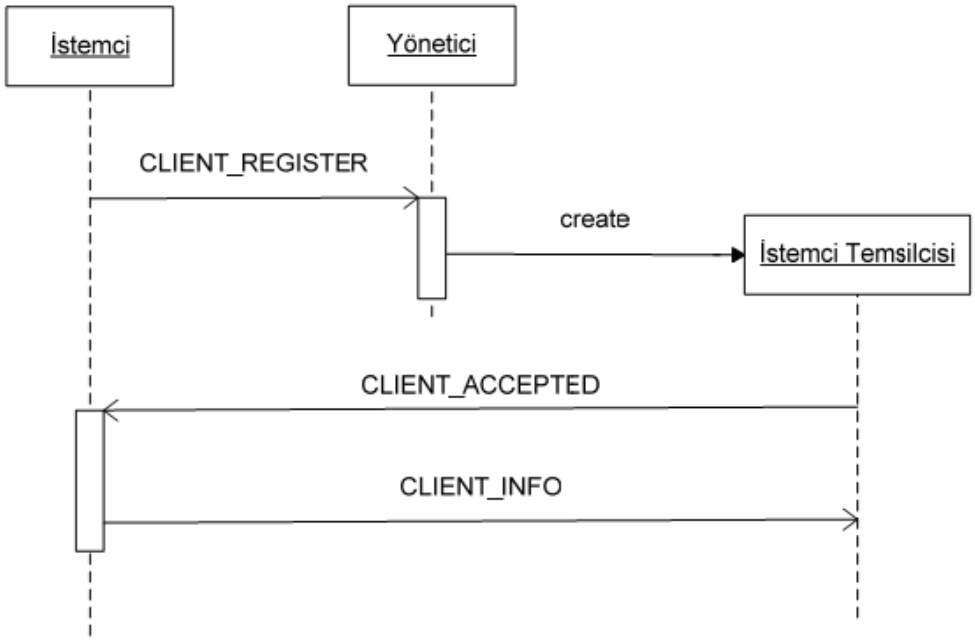
mesajı yollayarak bildirir.

2.Yönetici etmen bu mesajı aldıktan sonra bu istemcinin sisteme daha önce dâhil olup olmadığını araştırır. Eğer bu istemci sisteme ilk defa dâhil oluyorsa istemci için bir temsilci etmen yaratır ve istemci ve temsilcisinin kayıtlarını tutar.

3.Yaratılan bu temsilci istemci etmene CLIENT_ACCEPTED mesajı yollar.

4. İstemci bumesajı alınca temsilcisine kendisine ilişkin bilgileri içerik olarak barındıran CLIENT_INFO türünden bir mesaj yollar. Böylece istemci ve temsilcisi eşleştirilmiş olur ve istemci etmenin grid içindeki yaşam döngüsü başlar.

Şekil 2.3 :'de istemci etmenin sisteme bağlanışı sırasında oluşan mesajlaşma sırasını ve aktörlerini görebilirsiniz. Bu şekilde de görüldüğü gibi istemci etmen son aşamada kendisine ait bilgileri CLIENT_INFO mesajıyla temsilcisine bildirir. Bu mesaj türünün içeriği client.ClientInfo türünden bir nesnedir İstemci kendisi hakkındaki bilgileri ve yürütülmesini istediği görevleri bu mesajla temsilcisine gönderir [1].




**Şekil 2.3 :** İstemci etmenin sisteme bağlanma prosedürü.

Gönderilen mesajın içeriği istemciye ait ClientInfo türünden bir mesajdır. ClientInfo sınıfı java.io.Serializable sınıfından türetildiği için JADE ile gönderilen mesajların içeriği olarak kullanılabilmektedir. Böylece temsilci yönetici etmenle çalışarak bu işleri uygun işçilere gönderir [1].

**İşçi etmenin sisteme bağlanma protokolü**

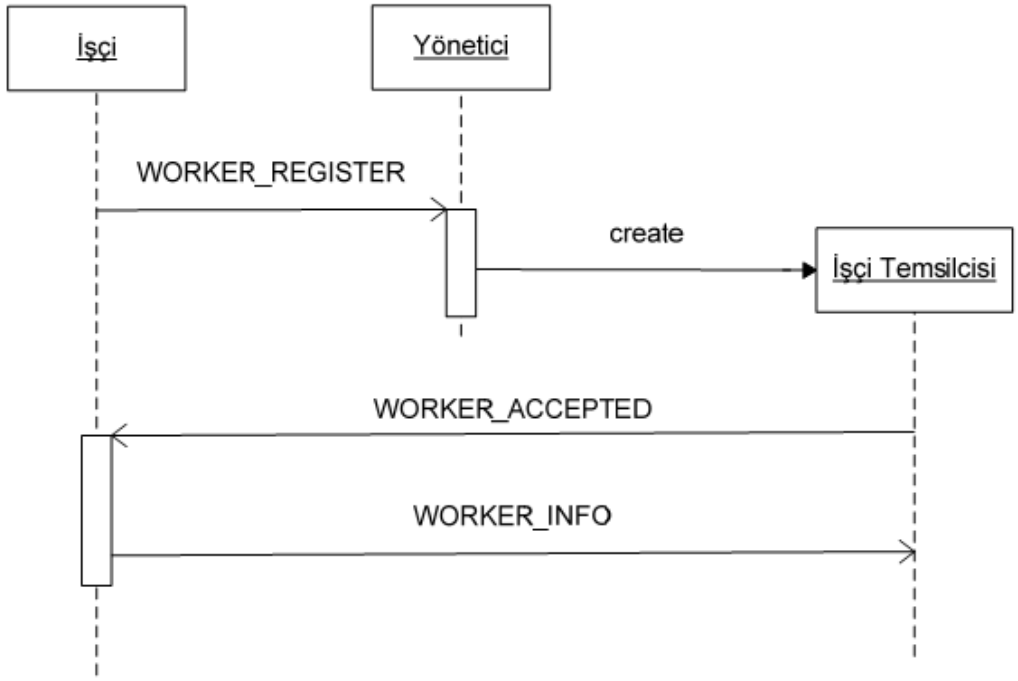
İşçi etmenler istemci etmenlere benzer bir protokolle bağlanırlar. Ancak farklı olarak işçi etmenlerin sisteme tekrar bağlanma durumu söz konusu olmamasından ötürü yönetici etmen her bağlanan işçi etmen için bir temsilci etmen yaratır [1].

Protokolün akışı şu şekildedir;

1.İşçi etmen sisteme bağlanma isteğini yönetici etmene WORKER_REGISTER mesajı yollayarak bildirir.

2.Yönetici etmen bu mesajı aldıktan sonra bir işçi temsilci etmen yaratır ve işçi ve temsilcisinin kayıtlarını tutar. Yaratılan bu temsilci işçi etmene WORKER_ACCEPTED mesajı yollar.

3.İşçi etmen bu mesajı alınca temsilcisine kendisine ilişkin bilgileri içerik olarak barındıran WORKER_INFO türünden bir mesaj yollar. Böylece işçi ve temsilcisi eşleştirilmiş olur.




**Şekil 2.4 :** İşçi etmenin sisteme bağlanma prosedürü.

Şekil 2.4 :'te [1] işçi etmenin sisteme bağlanışı sırasında oluşan mesajlaşma sırasını ve aktörlerini görülebilir. İşçi etmenin yolladığı WORKER_INFO mesajının içeriği worker.WorkerInfo türünden bir nesnedir. Bu mesajla işçi etmen temsilcisine sahip olduğu nitelikleri bildirir. Böylece işçi etmene görev atanması esnasında görevin

gerektirdiği özelliklere sahip olan işçi etmenin seçilmesi sağlanır [1].

#### 2.5.7.2 Görev atama protokolü

Grid sistemlerinde görevlerin uygun biçimde atanması ve olası hataların uygun bir şekilde giderilmesi hayatı bir önem taşımaktadır. Etmen tabanlı grid sisteminde uygulana protokol şu şekildedir [1].

1. Etmenler sisteme bağlanma protokolünde CLIENT_INFO mesajının içeriği olarak gönderdiği bilgilerden birisi de yürütülmesini istediği görev bilgileridir.

2. Temsilci etmen istemciye ilişkin bilgileri (ClientInfo) aldığında istemci tarafından gönderilen işleri de alır. İstemci temsilcisinde bir işçiye atanmayı bekleyen ve bir işçi tarafından yürütülmesine devam edilen işler listesi olmak üzere iki ayrı liste bulunur. Temsilci tüm görevleri bekleyen görevler listesine ekler.

3. Temsilci etmen belirli aralıklarla bekleyen işler listesini kontrol edip yönetici etmene görevin atanması için istekte bulunur. Belirli aralıklarla belirli işleri yinelemek için JADE tarafından sunulan jade.core.behaviours.TickerBehaviour sınıfı kullanılır [14]. Görevlerin kontrolü için bu sınıftan türetilmiş delegate.TaskScheduler sınıfı kullanılmıştır

4. Yönetici etmen sahip olduğu etmenler arasında meşgul olmayan ve özellikleri bu göreve uygun işçi etmenleri bulur.

5. Uygun bir işçi etmen varsa bu işçinin temsilci etmeni görevi TASK_REQUEST mesajıyla bu işçi etmene iletir.

6. İşçi etmen bu mesajı alınca görevi yürütmeye başlar.

7. İşçi temsilcisi görevin atandığını yönetici etmene bildirir

8. Yönetici etmen de görevin atandığını ilgili istemci temsilcisine bildirir.

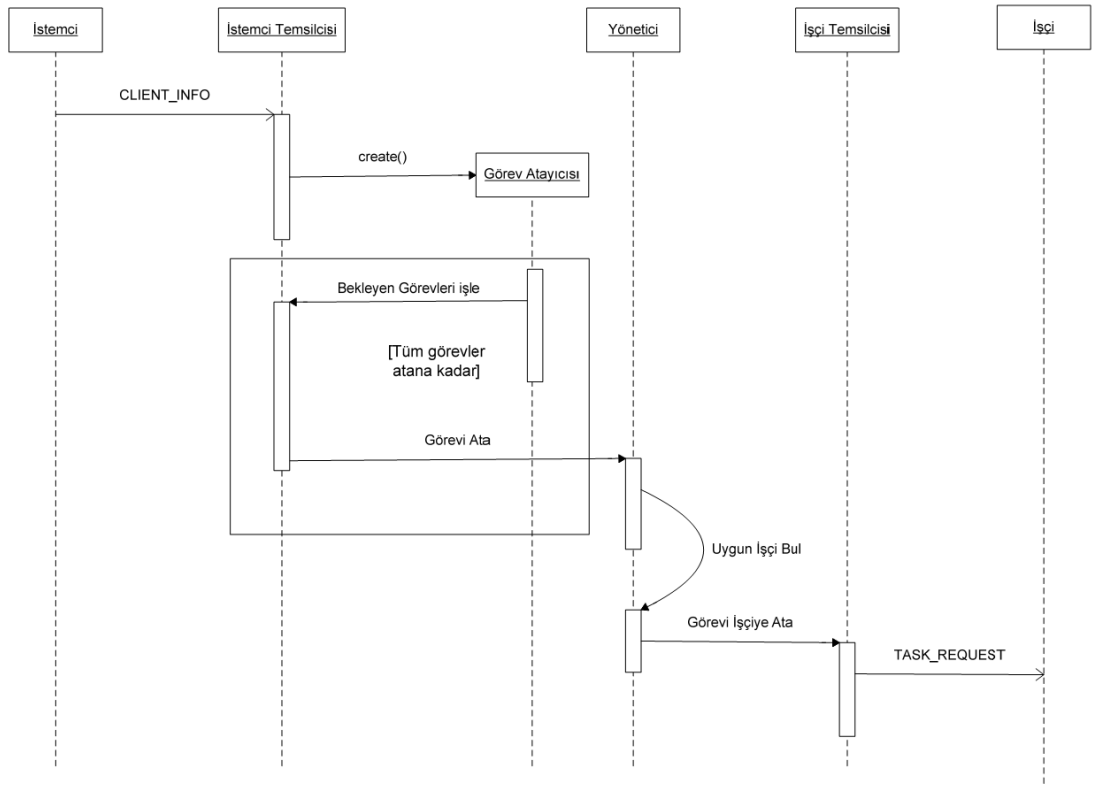
9. İstemci temsilcisi atanan görevi bekleyen görevler listesinden çıkartıp çalıştırılan görevler listesine ekler.

Görevlerin atanması protokolünde rol alan aktörleri ve mesajların akış sırası Şekil 2.5 :'te [1] verilmiştir.

#### 2.5.7.3 Etmenlerin sistemden ayrılma protokolleri

Etmen tabanlı grid sisteminde işçi ve istemci etmenlerinin sistemden çıkma

protokolü arasında bazı farklılıklar bulunmaktadır. İstemci etmenlerin sisteme bağlanıp görevleri temsilcisine ilettikten sonra sistemden çıkabileceği ve bir süre sonra tekrar sisteme bağlanabileceği düşünülmüştür. Ancak işçi etmen için böyle bir durum söz konusu değildir [1].




**Şekil 2.5 :** Görev atama prosedürü.

**İstemci etmenlerin sistemden ayrılması**

İstemci etmenler için iki çeşit sistemden ayrılma protokolü tanımlanmıştır. Birinci protokolde istemci sisteme yeniden bağlanmak üzere sistemden ayrılırken ikinci protokolde temelli sistemden ayrılmaktadır [1].

Aşağıda tekrar bağlanmak üzere ayrılma durumunda uygulanacak protokol verilmiştir [1].

1. İstemci temsilcisi istemci etmenin sistemde çıktığına karar verir. Bunun için aşağıdaki olaylardan birisi gerçekleşir.

a. İstemci temsilci etmene CLIENT_UNREGISTER mesajı yollar.

b. İstemci temsilci etmeni belirtilen süre boyunca istemci etmenden I_AM_ALIVE

mesajı almamıştır.

2. İstemci temsilcisi bu noktadan itibaren istemciden gelen I_AM_ALIVE mesajını beklemekte kullandığı sınıfı yapısından çıkartır, böylece artık istemcinin sisteme bağlı olup olmadığını kontrol etmez.

İstemci etmen bir şekilde sistemden temelli çıkma kararı verebilir. Örneğin istemcinin tüm görevleri tamamlanıp, sonuçlarını alması durumunda ya da istemcinin gerçeklemesine göre kullanıcı ara yüzünden bu bağlamda bir mesaj alınması durumunda, istemcinin sistemden çıkması istenebilir. Bu durumda yürütülen protokol şu şekildedir

1. İstemci etmen temsilcisine CLIENT_DONE mesajı yollar.

2. Temsilci, istemci etmenin sistemden çıkmak istediğini algılar ve yönetici etmene durumu bildirir.

3. Yönetici etmen istemcinin atadığı görevleri yürüten işçi etmenlerin temsilcilerini bulur.

4. Yönetici etmen her işçi temsilcisinden çalışma işini sonlandırmasını ister (stopWorking)

5. İşçi temsilcisi temsil ettiği işçi etmenine STOP_WORKER mesajını yollar.

6. İşçi etmen görevi sonlandırır ve temsilcisine WORKER_STOPPED mesajı yollar, böylece işçi etmen yeni görevler alabilecektir.

7. Tüm görevleri yürüten işçi etmenler için 4. 5. ve 6. adımlar yinelenir.

8. Yönetici etmen istemci temsilcisini sonlandırır ve istemci etmene ve temsilcisine dair tuttuğu kayıtları siler.

Şekil 2.6 :'da [1] istemci etmenin sistemden temelli ayrılmasına ilişkin protokolün akışı göstermektedir.

#### 2.5.7.4 İstemci etmenlerin sisteme yeniden bağlanması

Etmen tabanlı grid sisteminde istemci etmenlerin yürütülmesini istedikleri görevleri temsilcisine bildirdikten sonra sistemden ayrılma ve belirli bir süre sonra sisteme yeniden bağlanabilmektedir. Yeniden bağlanma protokolünde istemci etmen kendisine dair bilgileri tekrar temsilcisine göndermez, çünkü ilk bağlanma protokolünde bu işlem zaten yapılmıştır. İstemci temsilcisi istemci etmene yeniden

bağlanma anına kadar alınan tüm sonuçları bildirir [1]. İlgili protokol şu şekildedir.

1.İstemci etmen yönetici etmene CLIENT_REGISTER mesajı yollar.

2. Yönetici etmen bu mesajı alınca istemcinin daha önce sisteme bağlanıp bağlanmadığını kontrol eder. Daha önce bağlandığı sonucuna varırsa bu istemci için yaratılan temsilci etmeni bulur.

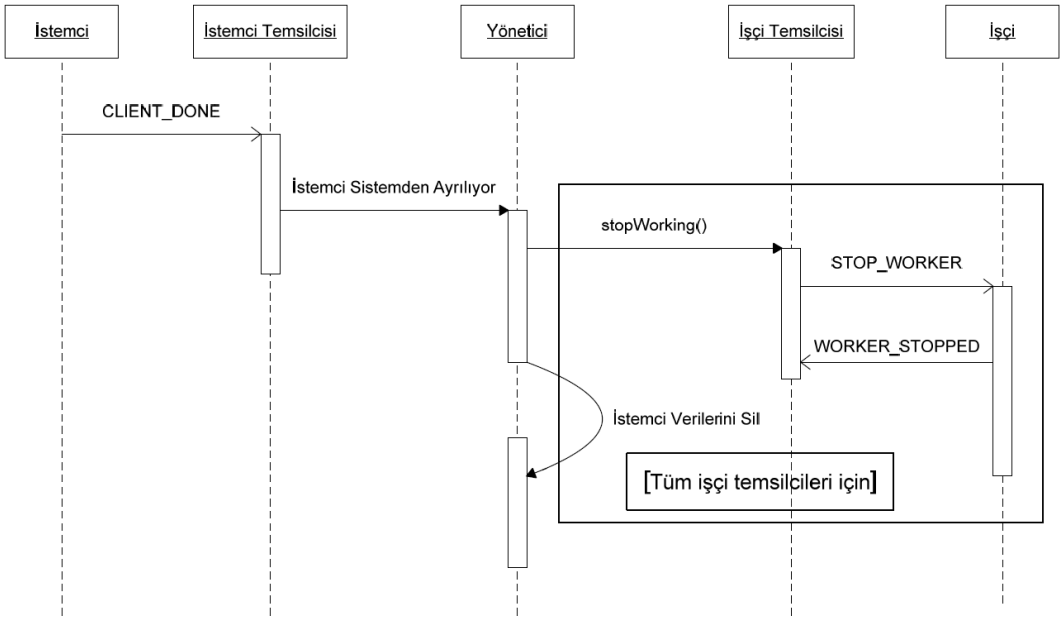
3. Bulunan temsilci etmene istemcisinin sisteme tekrar bağlandığını bildirir.

4. İstemci temsilcisi istemci etmene CLIENT_RECONNECT mesajını yollar. Bu mesajın içeriğine de o ana kadar tamamlanan görevlerin sonuçlarını ekler.

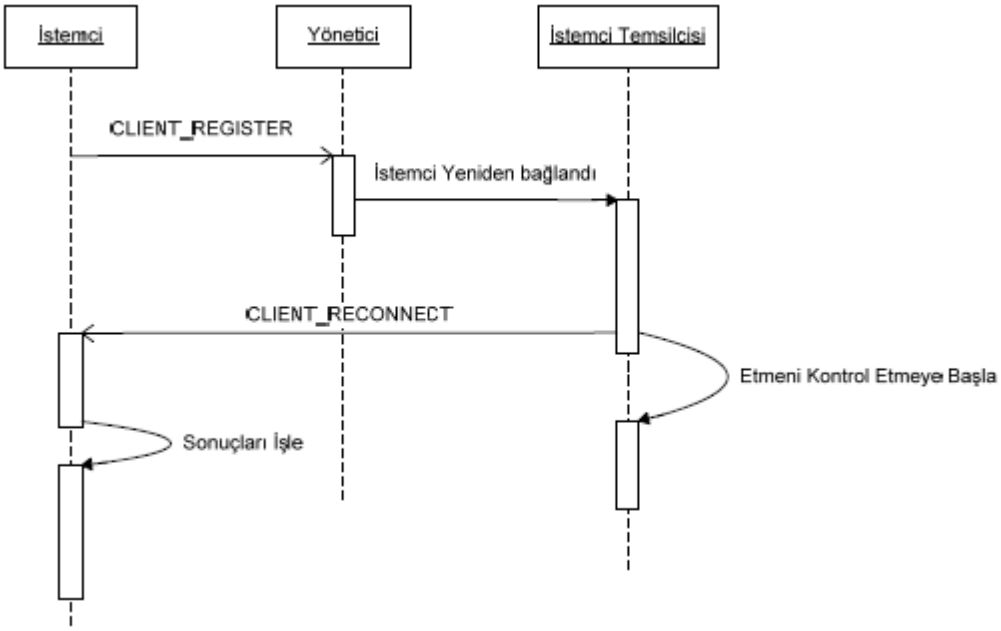
5. Aynı zamanda istemci etmenden I_AM_ALIVE mesajını beklemeye, yani tekrar istemcinin sistemden ayrılıp ayrılmayacağını düzenli olarak kontrol etmeye başlar.

6. İstemci etmen CLIENT_RECONNECT mesajını alınca mesaj içeriğindeki her bir sonucu işlemeye başlar.

Bu protokolün akışı Şekil 2.7 :'de [1] verilmiştir




**Şekil 2.6 :** İstemci etmenin sistemden ayrılması.




**Şekil 2.7 :** İstemci etmenin sisteme yeniden bağlanması.

# 3. GÜÇLÜ TAŞINABİLİRLİKLİ KOD GÖÇÜ

Bu bölümde güçlü taşınabilirlikli kod göçü ve bunun Java dili kullanılarak gerçeklenme yollarından bahsedilecektir. Daha sonra tezimizde kullandığımız Brakes [4] kütüphanesi tanıtılacaktır.

## 3.1 Güçlü Taşınabilirlikli Kod Göçü

Kod göçü bir uygulamanın çalıştığı bir makineden alınıp başka bir makinede çalışmasına devam edebilmesidir. Güçlü taşınabilirlik ise uygulamanın(yürütülen kod parçası) çalışma durumuna ait tüm bağlam bilgileriyle birlikte çalıştığı bir ortamdan alınıp, başka bir ortamda çalışmaya kaldığı yerden devam edebilmesi olarak özetlenebilir. Bir kod göçünün güçlü taşınabilirlikli sayılabilmesi için, uygulamanın çalışma durumunu içeren bağlam bilgilerinin göç edilen-hedef ortamda, programcı tarafından harici bir kaynaktan okunmasına ve yine programcı tarafından bu bilgilerle ilklendirmesine ihtiyaç duyulmadan, kaldığı yerden devam edebilmesi gerekir [16,17].

Tezimizde önerdiğimiz sistem etmen tabanlı bir grid altyapısında etmenler tarafından çalıştırılan güçlü taşınabilirlikli görevlerle ağ trafik yükü dengelemesinin gerçeklenmesidir. Çevresindeki diğer görevlerle sürekli etkileşim içersinde olan bir görev, içerisinde tuttuğu verileri bu etkileşim süresince sürekli olarak günceller. Bir görev verdiği kararlar ve gönderdikleri mesajlarla başka görevlerin verdiği kararları da etkiler, bu etkileşimler sonucunda tüm sistemdeki görevlerde hesaplanan veriler ve görevlerin çalışma akışları bu etkileşimlere göre şekillenir. Örneğin bir görev mesajlaştığı görevlerden gelen mesajlarla içeriğindeki bir değişkeni değiştiriyor olsun. Bir etmenden aldığı mesajla değişkene 5 eklerken, diğer bir etmenden gelen mesajla 2'ye çarpıyor olsun. Gönderdiği mesajlarla kendisine mesaj gönderen etmenleri de etkilediğini düşünelim. Mesajlaşmalar devam ederken tutulan değişkenin değerine göre de görevin çalıştırdığı program farklı dallanmalar yaparak çalışıyor olsun. Böyle bir durumda göç ettirilen görevin gittiği yerde aynı değerleri üreterek çalışmasına devam etmesine imkan yoktur. Çünkü gelen ve gönderilen

mesajların sırasının ve görevlerin çevresiyle etkileşiminin görev içerisinde tutulan değişkenin değerinde ve programın akışında doğrudan etkisi olmaktadır. Bu tip bir durumda görevin o ana kadarki tüm mesajlaşmaları ve etkileşimleri yeniden sağlanamayacağından, sağlansa bile mesajlaşılan etmenler canlı bir sistemde olmalarından ötürü zamana ve çevreleriyle etkileşimlerine göre anlık olarak farklı davranabileceklerinden, göç edilen yerde görevin değişkenlerini eski değerlerine ve akışını eski konumuna getirebilmesi için sistemin geçmişe yönelik işletilmesi mümkün değildir. Böyle bir sistemde yük dengelemesi gerçekleştirilirken bir göreve ait bağlam bilgisinin serileştirilerek tümüyle taşınması bir görevin o ana kadarki çalışmasının sonucu oluşturduğu verileri korumasının en iyi yoludur.

Java uygulamaları çalışma zamanında serileştirme için herhangi bir hizmet sağlamaz. Eğer çalışmakta olan bir uygulama, yalnızca nesne serileştirme ve sınıf yükleme mekanizmaları kullanarak yeni bir konuma göç ettirilirse, uygulamaya ait alt program çağrı yığını ve program sayacı verilerinden oluşan bağlam bilgisi kaybolur. Göç eden uygulamanın koduna ve verisine erişilebilir ancak uygulama çalışmaya kaldığı yerden değil de baştan başlar. Çalışma zamanındaki bir uygulamayı göç ettirdiğimizde o zamana kadarki bağlamının korunmuş olmasını ve göç ettiği yerde kaldığı yerden çalışmaya başlamasını bekleriz. Burada karşımıza çıkan sorun JVM'in bu yürütme bilgilerini programcılara vermemesi ve kendi içinde saklamasıdır [16]. Güçlü taşınabilirliğin sağlanması için yürütme bilgisine ihtiyaç vardır. Sonraki bölümde bu bilgilerin elde edilme yöntemlerinden bahsedilmiştir.

### 3.2 Java Dilinde Güçlü Taşınabilirlik Gerçeklenme Yolları

Tezimiz kapsamında kullanılan etmen tabanlı grid sistemi Java dilinde geliştirilmiştir. Bu sebeple tezimizin bir amacı olan güçlü taşınabilirliği grid üzerinde sağlamak için Java dili kullanılmıştır.

Güçlü taşınabilirlikli kod göçünün Java dilinde sağlanması için birçok yöntem geliştirilmiştir. Bu yöntemler standart sanal makine(VM) veya özelleştirilmiş/değiştirilmiş sanal makine kullanılma durumlarına göre iki temel sınıfta gruplandırılabilir [16].

JavaThread [18], D'Agents [19], Sumatra [20], Merpati [21] ve CIA [22] projelerinin çalışma prensipleri, standart JVM üzerinde eklentiler veya değişiklikler yaparak çalışan uygulamanın bağlam bilgisini JVM'den elde edilmesini sağlamaya

dayanmaktadır. Bu yolla elde edilen uygulamaya ait bağlam bilgisinin kullanılması için yine özelleştirilmiş JVM gerektiğinden, bu çözüm değiştirilmiş veya özelleştirilmiş JVM bağımlılığı getirmektedir. Bu yöntemin en önemli dezavantajı olarak taşınabilirliğinin zayıf olması olarak gösterilebilir. Var olan sanal makineler doğrudan kullanılamazlar ve Sun Java tarafından belirlenen tanımlamalara uyumlu halde tanımlanmış JVM'ler üzerinde çalışabilecek şekilde sürdürülmeleri de zordur. Örneğin JavaThread ve NOMADS JDK 1.2.2 ile uyumludur, D'Agents Java 1.0 VM'in modifiye edilmiş bir halini kullanır. Merpati ve Sumatra ise artık destek görmemektedir. Ayrıca JVM üzerinde yapılan değişiklikler ve eklentilerin getirdiği bir performans yükü de bulunmaktadır; bu sebeple bu yöntemlerde kullanılan özelleştirilmiş/değiştirilmiş sanal makineler ile Sun tarafından sunulan standart JVM'in performansını yakalamak zordur.

Java'ya güçlü taşınabilirliği eklemenin bir başka yolu da Wasp [23], JavaGo [24], Brakes [4], ve JavaGoX [25] projelerinde olduğu gibi derleme modelinde değişiklik yaparak, yürütme zamanından önce bağlam bilgisinin yakalanıp, yeniden kurma/devam etme için gerekli bazı komutların uygulama koduna eklenmesini sağlayacak bir önişlemden geçirilmesi işlemidir. Önişlemden geçirilen kod Wasp ve JavaGo projelerindeki gibi Java kaynak kodu olabileceği gibi, Brakes ve JavaGoX projelerinde olduğu gibi bytecode seviyesinde de olabilir. Uygulama kodunda yapılan değişiklikler sonucu oluşan sınıfların tüm Java ortamlarına uyumlu, ortamlar arasında taşınabilir ve platform bağımsız oluşu bu yaklaşımın en önemli avantajı olarak gösterilebilir [16]. Ayrıca bytecode gerektiğinde Java sınıf yükleme mekanizmaları tarafında ağ üzerinde kolaylıkla indirilip kullanılabilir bir yapı sunar [4].

Tez çalışması kapsamında [16]'da önerilen sisteme benzer bir sistem kullanılmıştır. [16]'da önerilen sistemde Brakes kullanılarak içeriğine bağlam bilgisi eklenmiş bir Java ipliği(thread) yani uygulama kodu serileştirilerek hedef sisteme(JVM) kaldığı yerden çalıştırılmak üzere Java RMI kullanılarak gönderilmektedir. Tezimizde ise yine Brakes kullanarak bağlam bilgisiyle birlikte serileştirdiğimiz Java ipliğini bölüm 2.4.4'de bahsettiğimiz JADE ACL mesajlarının serileştirilebilir Java nesnelerini kabul eden içerik(context) kısmına ekleyerek bir etmenden diğerine gönderilmesi sağlanmıştır.

## 3.3 BRAKES

Brakes Java ipliğine ait bağlam bilgisinin kaydedilip yeniden kurulmasını sağlayan bir mekanizma sunar [4,16]. Brakes temelde iki parçadan oluşur:

1. Java sınıflarının kendi bağlam bilgilerini istedikleri zamanda yakalayıp kaydetmelerini sağlayacak bir bytecode dönüştürücüsü.
2. Sınıflara eklenmiş bağlam bilgilerini kullanarak ipliklerin istenilen zamanda duraklatma ve devam ettirme işlemlerini gerçekleştirmelerini sağlayan bir altyapı.

Brakes, Katholieke Universiteit Leuven Belçika'da, dağıtılmış sistemler ve bilgisayar ağları (DistriNet [28]) araştırma grubu tarafından akademik açık lisans (AFL) altında geliştirilmiştir.

Bir Java ipliğinin göç etmesinin sağlanması için ipliğin çalıştırılmasının duraklatılıp, yığın bilgileri ve program sayacı değerinin elde edilerek serileştirilip, çalışmaya devam edilecek olan hedef sisteme gönderilebilir hale getirilmesi gerekir. Hedef sistemde ise serileştirilmiş yığın bilgileri ve program sayacı yeniden kurularak, iplik çalıştırılmaya kaldığı yerden devam etmeye hazır hale getirilmelidir [16]. Bu düzende gerçekleştirilen göç etme işlemi ipliğin "güçlü taşınabilirlikli göçü" olarak tanımlanır.

## 3.4 Java İpliklerinin Serileştirilmesi

İplik serileştirilmesi mekanizması çalışan ipliğe ait durum (bağlam) bilgilerinin elde edilmesiyle gerçeklenmektedir. Bir bytecode dönüştürücü tarafından uygulamanın çalışma anındaki bağlam bilgilerini saklayıp başka bir ortamda yeniden kurulmasını sağlayacak kodlar uygulama koduna eklenir. Böylece, uygulama kodu değiştirilerek ipliğin bağlam bilgisi bytecode seviyesinde eklenmiş olur. Bu mekanizma sayesinde gerçekleştirilen göç standart JVM'ler tarafından çalıştırılabilir olmaktadır [26].

Brakes, serileştirme mekanizmasının göç için uygun bir yapıda olmasını sağlamak üzere JVM ipliklerini tamamlayıcı ek bir görev(task) yapısı sunmaktadır. JVM iplikleri bu görevler tarafından sarmalanarak, ipliklerin ve görevlerin bağlam bilgileri ortak hale gelmektedir [26].

### 3.4.1 Serileştirilebilir Görevler

Görev serileştirilmesi her çalışma noktasında mümkündür [26]. Ancak bu görevlerin istenilen her anda serileştirilebilir olduğu anlamına gelmemektedir. Görevler ancak çalışma durumuna ait bilgilerin yakalanması işleminin tamamlanmasından sonraki ve işlemin yakalanan çalışma durumuna yeniden kurulmasından önceki noktalarda serileştirilebilirler [4].

Yakalanan çalışma durumu, yani bağlam bilgileri görev içerisinde tanımlanmış bir içerik(context) nesnesinde tutulur [4]. Bu içerik nesnesi Java nesne serileştirme mekanizmasıyla serileştirilebilir bir yapıdadır. Yakalanan/kaydedilen bağlam bilgisi daha önce de bahsedildiği gibi programa ait o anki Java yığınından ve son işletilen komutun yerinden yani program sayacının değerinden oluşur. Bir görevin içerisindeki ipliğin yakalanıp yeniden kurulabilmesi için uygulama koduna bytecode seviyesinde komutlar eklenmelidir [26].

Ayrıca bir görevin hangi çalışma modunda olduğunu belirten çeşitli bayraklar da bulunmaktadır [4]. Bir görev üç farklı çalışma durumunda bulunabilir:

- Çalışıyor (running): görevin normal çalışma durumu.
- Yakalama (capturing):bu durumda görevin o anki bağlam bilgileri içerik nesnesine aktarılır. Bu durumdayken `isCapturing` bayrağı kurulur.
- Yeniden Kurma (restoring): bu durumda görev içerikte kayıtlı bağlam bilgisi kullanılarak son çalışma durumuna getirilir. Bu durumdayken `isRestoring` bayrağı kurulur.

Serileştirilmiş görevlerin yeniden kurulması işlemi Java nesne serileştirme işlemiyle benzer şekilde otomatik bir işlem değildir, başlatılmaya ihtiyaç duyar. Başlatılma işlemi görev tarafında iki metod şeklinde gerçeklenmiş iki özel prosedürün çağrılmasıyla gerçekleştirilir. Bu özel prosedürler bağlam bilgisinin yakalanması ve yeniden kurulmasını sağlamakla görevlidirler [4].

- `public void yield()`: Görev serileştirme mekanizması bağlam bilgisinin göç edecek görevin çalışması sırasında yakalanmasına ihtiyaç duyar. yield() metodu görev duraklatılmak istendiğinde çağrılır, göreve ait bağlam bilgisi serileştirilmiş bir formatta yakalanır [4].
- • `public void resume()`: Bu metod çağrılarak görevin görev

sıralayıcı (task scheduler) tarafından çalıştırılmasının planlanması sağlanır. Sıralayıcı görevin çalıştırılma zamanı geldiğinde, önce görevin bağlam bilgisi yeniden kurulur, sonra görev kaldığı yerden çalışmaya devam eder. Bu metod göç eden görevin iletilmesiyle birlikte hedef sistemde çağrılır [4].

Görevlerin çalıştırma planlaması görev sıralayıcısı tarafından gerçekleştirilir. Görev sıralayıcısı görevlerin çalıştırılma sıralarını düzenler. Sıralayıcı bir görevi başlattığında görevin çalışmasının sağlanması için görev bir JVM ipliğine atanır. Her bir görev için bir iplik yaratılıp sisteme dahil olmasına izin verilerek, görev işletimi sırasında farklı ipliklerin birlikte çalışması sayesinde, JVM tarafından sağlanan içerik değiştirmesi(context-switching) hizmetinden faydalanılır [4].

### 3.4.2 Bağlam Bilgisi Yakalama

Bir görev yakalama durumuna geldiğinde, o anki çalıştırılan metoda ait yığın verileri ve en son yürütülmüş olan komut (LPI-last performed instruction) kullanılarak oluşturulan yapay program sayacı kaydedilerek sonlanır. Brakes yığın içerisinde ilerleyerek her bir metod için bu işlemi tekrarlar [4]. Bağlam bilgisinin kaydedilmek istendiği metoda gelindiğinde Brakes metodlar içerisindeki ilerlemesini bitirir. Bu işlem Şekil 3.1'de gösterilmiştir. Yığın içerisindeki programın akışına dahil olan metodlar için bağlam bilgisi içerik verisine kaydedilmiş olur.

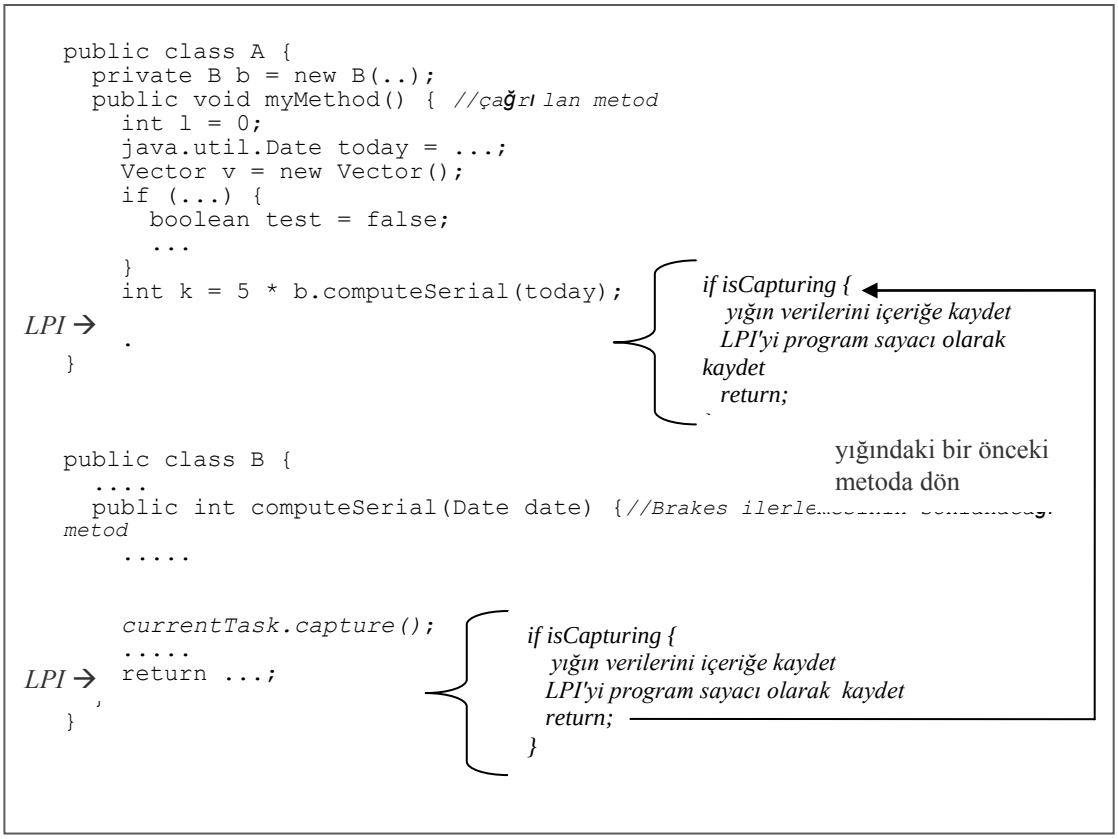
### 3.4.3 Bağlam Bilgisi Yeniden Kurma

Bir görev Şekil 3.2 'de gösterildiği gibi yeniden kurulmak istendiğinde, görev tarafından yeni bir iplik başlatılır. Görev nesnesinin resume() metodu çağrılarak, görev sıralayıcı tarafından yeniden planlanır. Yeniden kurma işlemi aslında görev sıralayıcı tarafından görevin çalıştırılmaya başlandığında başlar [4]. Öncelikle yığın verileri kurulur, ve LPI (last performed instruction- son yürütülen komut) kontrol edilerek görevin son olarak hangi metodda kaldığı belirlenir. Her metod bir sonraki metoda ait yığın verilerini yükleyerek ve LPI kontrol ederek o metoda dallanır. Bu şekilde ilerlenerek iplik bağlam bilgisi yakalanmış metoda ulaşır, yığın verilerini yükler ve program sayacını kurar. Yeniden kurma bayrağının işareti kaldırılarak görev kaldığı yerden çalışmasına devam eder [4].

### 3.4.4 ByteCode Dönüştürücüsü

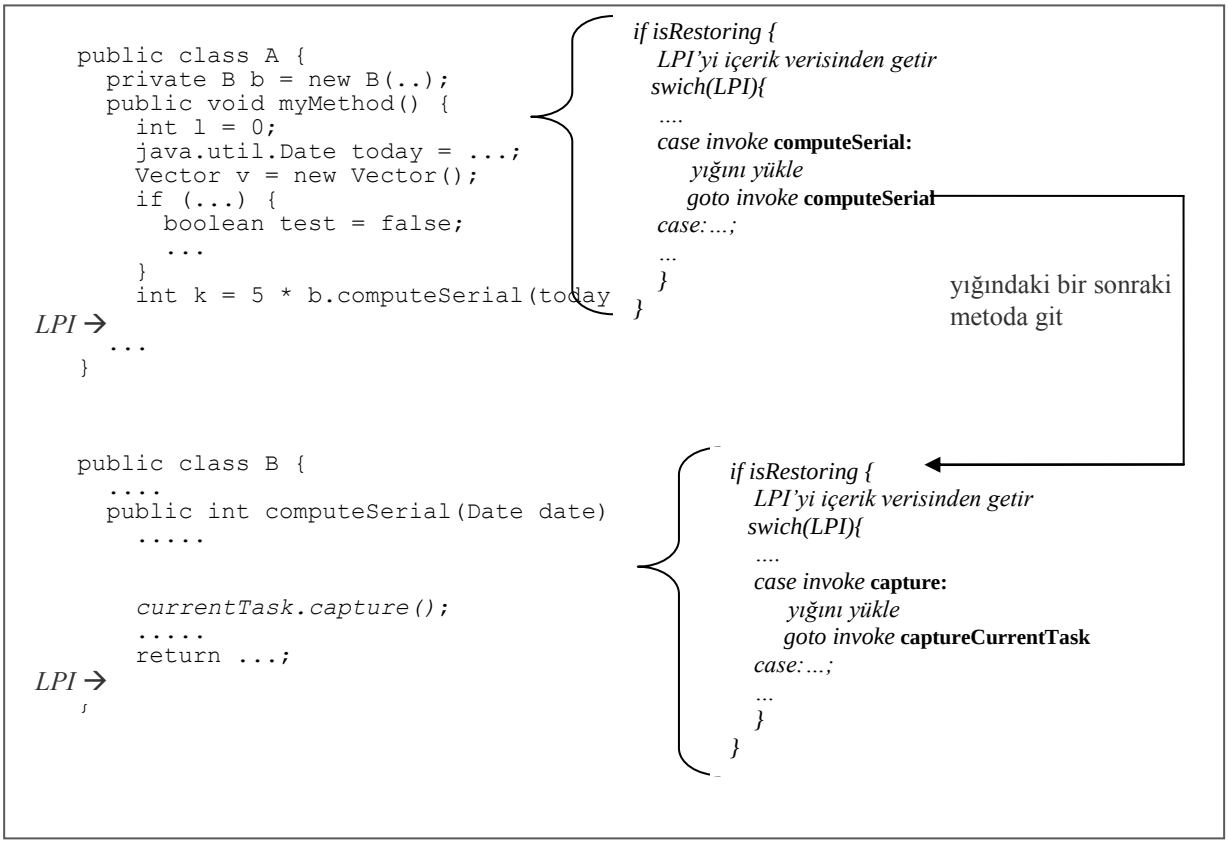
Bir metod içinde kalınan komuttan devam edecek şekilde dallanmayı sağlayacak kod

bloğunun uygulamaya ait bytecode verisine doğru bir şekilde eklenmesini sağlayabilmek için, öncelikle metodun duraklatıldığı komuta kadarki kısımda yığınında bulunan verilerin bilinmesi gereklidir. Metodun duraklatılan komuta kadar kullandığı yerel değişkenlerin ve yığın içerisinde yer alan verilerin tiplerinin analizi gereklidir. Verilerin tiplerine ait bilgiler bytecode içersinde belirtilmediğinden, bu analizler oldukça karmaşıktır. Brakes bu analizin gerçeklenmesinde Java bytecode doğrulayıcısının kullandığı tip çıkarımı yöntemine benzer bir yöntem kullanmaktadır [4].




**Şekil 3.1 :** Bağlam bilgisi yakalama

Tip çıkarımı analizi tamamlandıktan sonra kod dönüştürücü her metod için uygulama byte code'unu yeniden yazar. Metodlar içerisindeki her *invoke* çağrısı için yeni bir *invoke* çağrısı yazılmaktadır. Şekil 3.1 : ve Şekil 3.2 :'de görüldüğü üzere bytecode içerisindeki *invoke* komutları program içerisindeki dallanmaları düzenleyen çağrılardır. Tekrar yazım işlemi sonucunda program kodundaki dallanmalar yeniden ayarlanır. Duraklatılmış program devam ettirilmek istendiğinde, programın metodlar üzerinde dallanarak çalışmanın kaldığı yere gelmesi ve buradan devam etmesi sağlanır.




**Şekil 3.2 :** Bağlam bilgisi yeniden kurma

# 4. YÜK DENGELEME

Bu bölümde etmen tabanlı grid sistemi üzerinde gerçekleştirilecek olan yük dengeleme işleminden ve yük dengeleme yapılırken kullanılan görev göçüne karar veren algoritmadan bahsedilecektir.

## 4.1 Yük Dengeleme

Yük dengeleme herhangi bir sistemde paylaştırılan ve ortak kullanılan her türlü kaynak üzerinde oluşan veya oluşabilecek talep yoğunluğunun giderilmesi veya önlenmesi olarak açıklanabilir. Etkin bir kaynak paylaşımı gerektiren dağıtılmış yapıları nedeniyle grid sistemleri için yük dengeleme temel bir ihtiyaçtır. Grid sistemleri kendisine dahil olan makinelerdeki hizmetlerini kaliteli bir şekilde ve düşük tepki süresiyle sunmayı amaç ederler. Bunu sağlamalarının en kolay yolu ise sistem üzerinde yapılacak yük dengelemeleridir.

Etmen sistemlerini ele alacak olursak, etmenler değişken çevre koşullarını algılayıp, çeşitli kararlar vererek değişen çevre koşullarına hızlı bir şekilde cevap verme yeteneğine sahiptirler. Ayrıca etmenler sosyal ve interaktif bir şekilde çevreleriyle ve birbirleriyle iletişim halindedirler. Etmenlerin bu özellikleri göz önüne alındığında kısıtları olan bir sistemde, örneğin CPU kaynağının kısıtlı olduğu bir ortamda, etmenler bu kısıtı algılayıp birbirleriyle iletişime geçerek birlikte veya tek başlarına ilgili kısıtın etkilerini azaltabilecek kararları alabilirler. Etmenlerin bu esnek algılama, haberleşme ve karar verme mekanizmaları sayesinde değişken çevre koşulları sonucunda alacakları kararlarla yük dengelemeye çok yatkın bir yapıları vardır.

Yük dengeleme grid sistemlerinin doğasında olan bir özellik olarak görülebilir. Grid sistemlerinde bir özellikten öte bir gereklilik olarak görülebilecek yük dengeleme mekanizmalarına etmenlerin interaktif ve otonom yapılarının getirdiği yük dengelemeye uygun yapılarının eklenmesinin sonucu olarak etmen tabanlı bir grid sisteminde yük dengelemeleri gerçeklenmesi doğal karşılanmalıdır. Tezimizde etmen tabanlı grid sisteminin bu özellikleri göz önünde bulundurularak ağ trafik yükünü

dengeleyen bir algoritma geliştirilmiştir.

### 4.2 Kısıt Karşılama Problemleri

Bir kısıt karşılama problemi değer ataması yapılması beklenen bir grup değişkenle ilgili atamaların nasıl yapılacağının belirlendiği bir grup kısıttan oluşmaktadır. Ayrıca her bir değişken için atama yapılmasının mümkün olacağı değerler kümesi bulunmalıdır. Bir kısıt karşılama probleminin çözümü, çeşitli arama metodları kullanılarak, değişkenler kümesinde tanımlı değişkenlere kısıtlar tarafından belirlenen sınırlar içerisinde kalma koşuluyla, değerler kümesinden uygun atamaların yapılmasının sağlanmasıdır. Bazı durumlarda bir fonksiyon tarafından kısıtların karşılandığı yani tüm değişkenlere uygun birer değer atamasının yapıldığı çözüm kümeleri için maliyet hesabı yapılarak ideal değişken-değer atamalarının yapılması sağlanabilmektedir [27].

Kısıt karşılama problemlerine verilebilecek açıklayıcı basit bir örnek [27]'de verilen harita boyama örneğidir. Bu örnekte birbirine sınır olan komşuların farklı renklerde boyanmaları kısıt olarak tanımlanmıştır, değişkenler atama yapılması istenen yedi ülke ve değerler kümesi ise kırmızı, yeşil ve mavi renkleri olarak belirlenmiştir. Komşu ülkeler farklı renklerde boyanacak şekilde haritadaki ülkelere renk ataması yapılması ve bu bağlamda değişken-değer yani ülke-renk çiftlerinin belirlenerek bu atamaların kısıta olan uygunluklarının bir arama algoritmasıyla denenmesi kısıt karşılama problemlerinin ve çözümlerinin bir örneği olarak verilebilir [27].

Bilgisayar bilimlerinde özellikle yapay zeka alanında bir çok problem kısıt karşılama problemi şeklinde formüle edilerek çözülebilmektedir. Tezimizde mesajlaşma sayıları, etmenlerin bağımlılıkları, göç maliyeti ve bir taşıyıcıda bulunabilecek etmen sayısı gibi parametreleri kısıt olarak tanımlayarak, göç karar algoritmasını kısıt karşılama problemi olarak formüle ettik. Göç karar algoritmamız olarak nitelendirebileceğimiz kısıt karşılama probleminin çözümünde açık kaynak kodlu ve Java tabanlı bir kısıt çözüm motoru olan Jacop kütüphanesini [5] kullandık.

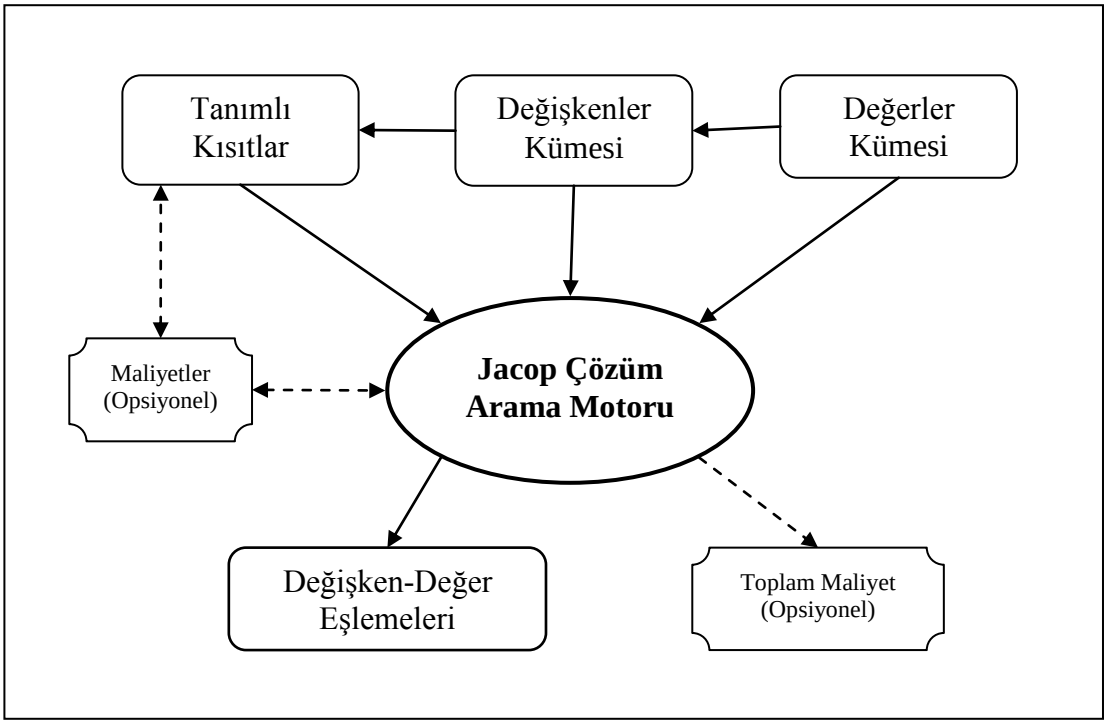
#### 4.2.1 Jacop

Jacop(Java Constraint Programming solver) kısıt programlama paradigmalarının Java dili kullanılarak gerçeklendiği bir kütüphanedir. Jacop kısıt karşılama problemlerinin temelinde yer alan değişkenlerin, değişkenlere atanacak değerlerin ve değişkenlerle

kısıtların tanımlanmasını sağlamasının yanında kısıtları tanımlanmış bir problemin çözümünü bulmada kullanılacak arama metotlarının belirlenmesine de imkân tanır [5].

Jacop kütüphanesi eşitlik, eşitsizlik, mantıksal, koşula bağlı gibi temel kısıtların kolaylıkla tanımlanabilmesini sağlar. Bu temel kısıtların yanında Jacop tarafından global ve ayrıştırma kısıtları şeklinde tanımlanan, belli kısıt paradigmalarının hazır olarak gerçeklenmiş olarak sunulduğu daha karmaşık kısıtlar da Jacop tarafından kullanıcılara sunulmaktadır [5].

Şekil 4.1 :'de Jacop kütüphanesinin işleyişine ait genel mimari gösterilmektedir.




**Şekil 4.1 :** Jacop mimarisi.

Jacop kütüphanesinde tüm kısıtlar JaCoP.constraints.Constraint sınıfından türemiş nesnelerdir. Aşağıda tezimiz kapsamında kullandığımız birkaç kısıt ve açıklaması verilmiştir.

- IfThenElse(c1, c2, c3): Eğer c1 kısıtı karşılanıyorsa ise c2 eklenir değilse c3 eklenir.
- XeqY(X, Y) : X ve Y değişken olmak üzere X'in Y'ye olma kısıtı.
- XneqC(X, Const c) : X değişken , c bir sayı olmak üzere X'in c'ye eşit olmama kısıtı.

- XplusCeqZ(X, Const, Z) : X değişkeni ve bir sayının toplamının Z değişkenine eşit olması kısıtı.
- XplusYeqZ(X, Y, Z) : X ve Y değişkeninin toplamının Z değişkenine eşit olması kısıtı.
- Or(ArrayList<PrimitiveConstraint> c) : c kısıtlar listesi olmak üzere listedeki kısıtların mantıksal veya'lanması kısıtı.
- count(value, [x1, x2, . . . , xn], var) : var değişkenindeki liste içerisinde value değerinin kaç kez bulunduğunu sayar.

Problemle ilgili tüm kısıtlar, değişkenler ve değerler JaCoP.core.Store tipinden bir nesneye eklenir. Store sınıfı üzerinde çözüm aramalarının yapıldığı temel sınıftır. Tüm tanımlamalar yapıldığında çözüm araması yapılarak Jacop tarafından uygun değişken-değer eşleştirmeleri yapılır.

Tezimizde problem tanımlanırken dahil edilen kısıtlarda, kısıt karşılama problemlerinin bir kolu olan ağırlıklı kısıtlar (weighted constraints) kullanılmıştır. Bu tarz kısıtlar bize problemde bir kısıtın gerçekleşmesi durumunda sisteme getirebileceği maliyeti tanımlayabilme imkanı vermektedir. Jacop kütüphanesi tanımlı problem üzerinde çözüm aramaları yaparken kısıtlar için tanımlı maliyetleri göz önünde bulundurarak toplam maliyeti minimize edecek şekilde arama yapabilme yeteneğine sahiptir.

Jacop kısıt karşılama problemi şeklinde fomüle edilebilecek problemler için esnek bir çözüm yolu sunmaktadır. Tezimizde Jacop kütüphanesini kullanmamızın temel sebebi, Jacop kütüphanesinin doğru bir şekilde kısıt karşılama problemine dönüştürülmüş bir problemi hızlı ve optimal bir şekilde çözme yeteneğidir.

### 4.3 Ağ Trafik Yükü Dengeleme Algoritması

Etmen tabanlı grid sisteminde etmenler kapsayıcı(container) adı verilen yapılar üzerinde çalışmaktadır. Her bir kapsayıcı fiziksel olarak farklı bir makinede yer almaktadır. Kapsayıcılarda dağıtılmış olarak çalışan etmenler birbirleriyle mesajlaşan görevleri çalıştırmaktadırlar. Kapsayıcılar 5. Bölümde daha detaylı olarak ele alınacaktır. Tezimizde farklı kapsayıcılarda yer alan etmenler tarafından çalıştırılan ve birbirleriyle mesajlaşan görevler gerekli hesaplamalar yapıldıktan sonra uygun görülmesi halinde aynı kapsayıcıya taşınarak, kapsayıcılar arasında oluşan

mesajlaşma sayılarının azaltılması ve bu sayede kapsayıcılar arasındaki ağ trafiği yükünün dengelenmesi amaçlanmıştır.

Tezimiz kapsamında ağ trafik yükü dengeleme algoritmasının yani görev göçlerine karar veren algoritmanın kısıt karşılama problemi şeklinde formüle edildiğinden bahsetmiştik. Problem tanımlaması ve çözümün Jacop tarafından bulunması işlemleri aşağıdaki adımlarla gerçekleştirilmiştir:

1. Her göç hesaplaması için öncelikle bir tane JaCoP.core.Store nesnesi yaratılmaktadır. Store nesnesi bir nevi veritabanı olarak düşünülebilir. Tüm değişkenler bu nesne kullanılarak bu veri tabanı içinde tanımlanır. Yine veritabanı örneğinden devam edersek kısıtlar da bu veritabanına kayıtlı sorgular olarak düşünülebilir. Yeni kısıtlar Store nesnesinin impose yordamı ile eklenmektedir.
2. Kısıt karşılama problemimizin değişken kümesi Store nesnesi kullanılarak yaratılır. Tezimiz kapsamında değişken kümesi görevlerimizi temsil eder.
3. Değişkenler kümesi tanımlanırken her bir değişkenin alabileceği değerler kümesi de belirlenir. Tezimiz kapsamında değerler kümesi kapsayıcılardan oluşmaktadır. Bir görevin çalıştığı kapsayıcı, mesajlaşma işlemleri açısından önem arzetmektedir. Bir görevin başka bir kapsayıcıya gönderilmesi, kısıt karşılama problemi olarak düşündüğümüzde bir değişkene yeni bir değer atanması, Jacop tarafından gerçekleştirilecek çözüm işleminden beklediğimiz sonuçtur. Hesaplamalar sonucunda değişkenlere yeni değerler atanması, görevin belirlenen kapsayıcıya taşınması gerektiği kararı verilmesi demektir.
4. Görevler arası mesajlaşmalar için koşullu(IfThenElse) kısıtlar yardımıyla mesajlaşmalar için problem tanımına maliyetler eklenir. Eğer mesajlaşan görevler aynı kapsayıcıda çalışıyorlarsa daha az bir maliyet eklenir. Tezimizde aynı kapsayıcıda çalışan görevlerin mesajlaşmalarının sisteme bir maliyeti olmadığı varsayılmıştır. Farklı kaspayıcıdaki görevlerin mesajlaşmalar sayıları maliyet olarak eklenmiştir.
5. Görevlerin taşınması durumunda oluşacak maliyetler sisteme eklenir. Eğer sistemdeki bir görevin çalıştığı kapsayıcı değişecekse bu durumla ilgili bir kısıt ve taşıma maliyeti sisteme eklenir. Bu kısıt tanımlanırken yine koşullu kısıtlar kullanılmıştır.

6. Görevlerin bağımlılıkları bulunduğu kapsayıcılarda kalmaları kısıtları tanımlanır. Görevler talep ettiği özellikleri barındıran kapsayıcılar altında çalışmalıdırlar. Örneğin özel bir donanım gereksinimi olan bir görev o donanıma sahip makinelerde çalışabilmektedirler.
7. Kapsayıcılarda çalışabilecek maksimum ve minimum sayıdaki görevler kısıt olarak belirlenir. Tezimizde belirlediğimiz kapsayıcıdaki çalışabilecek minimum görev sayısı sıfır iken maksimum sayı ilgili kapsayıcıda yer alan işçi etmen sayısı kadardır.
8. Taşıma maliyetleri ile mesajlaşma maliyetlerinin toplamı bir değişkene atanmak üzere kısıt tanımlanır. Bu tanımlanan değişken yapılacak arama işleminde maliyetin minimum olduğu çözümün aranmasında kullanılacaktır.
9. Son olarak Store nesnesine eklenen değişken ve kısıtlar üzerinde yapılacak bir arama ile gerekli değişken-değer eşlemeleri elde edilir. Bu işlem Store nesnesini veritabanına benzetirsek veritabanındaki tüm sorguların çalıştırılıp sonuçların alınmasına benzetilebilir. Jacop tarafından arama işlemi toplam maaliyet olarak tanımlanan değişkeni en düşük yapacak şekilde işletilir.

İşletilen aramadan sonra bulunan sonuçlar neticesinde göç yönetici etmen tarafından görev göçleri için gerekli işlemler yapılır. Göç yönetici etmen yukarda tanımlanan adımlardan oluşan prosedürü belli aralıklarla işleterek ağ trafiğini dengeleme işlemini devam ettirir.

# 5. ÖNERİLEN SİSTEM

Bu bölümde etmen tabanlı grid sistemi üzerinde görevler arasındaki mesajlaşma yoğunlukları göz önünde bulundurularak güçlü görev göçüne dayalı ağ trafik yükü dengeleme algoritmasının gerçeklenme detayları anlatılmıştır.

## 5.1 Sistemin Analizi

Tez çalışması kapsamında tasarlanan sistem, farklı makinelerde çalışırken birbirleriyle mesajlaşan görevlerin aynı makineye taşınarak mesaj trafiğinin neden olduğu ağ yükünün azaltılmasını hedeflemektedir. Taşıma işleminin gerçekleştirilmesine karar veren algoritma görevlerin bağımlılıklarını ve göç ettirme işleminin sisteme getireceği taşıma maliyetlerini de hesaba katarak tüm verileri bir kısıt karşılama problemi haline getirmektedir. Kısıt karşılama probleminin çözümü sonucunda verilen göç kararları ile görevler belirlenen makinelere göç ettirilmektedir.

### 5.1.1 Bileşenlerin Sisteme Dahil Edilmesi

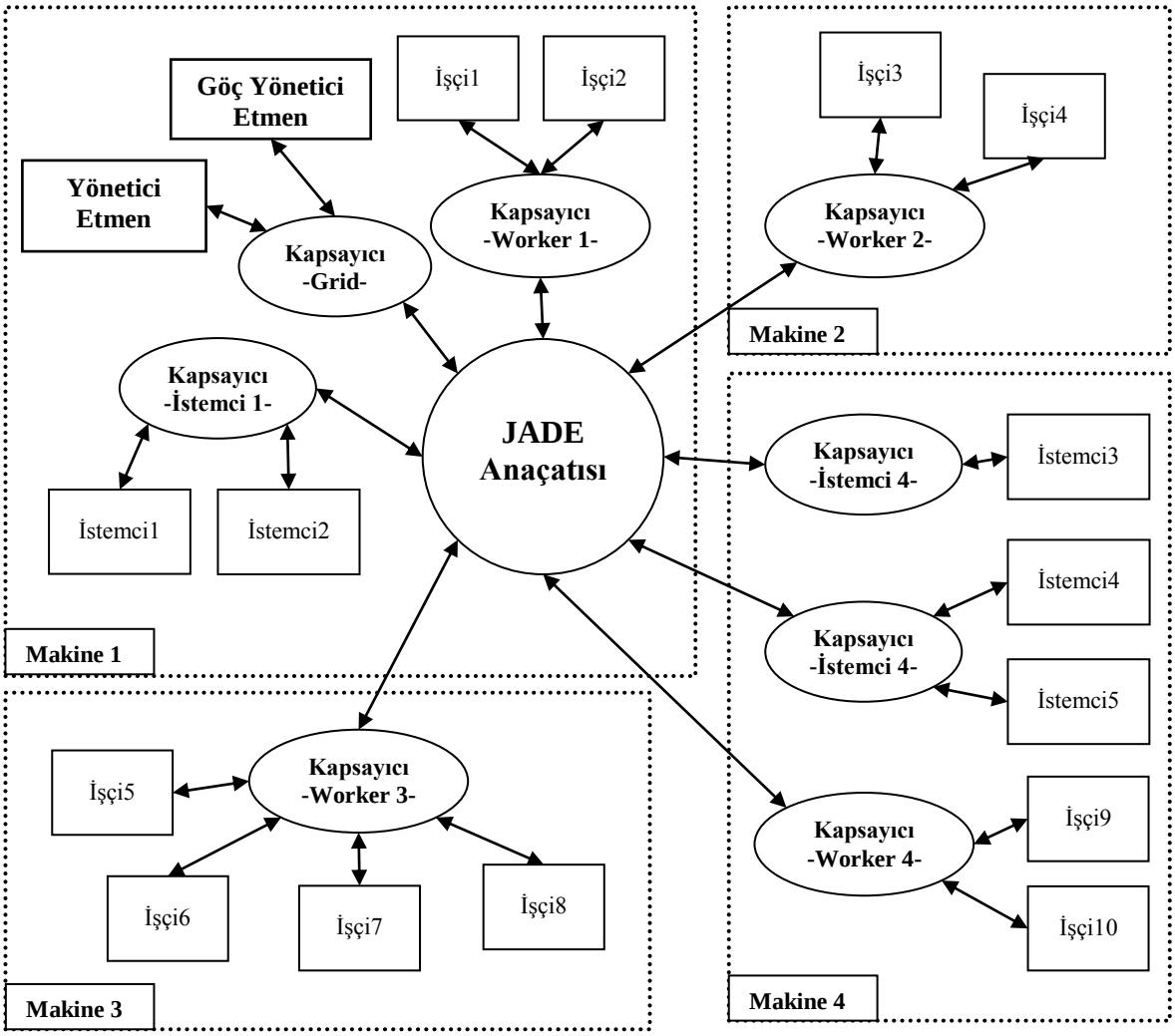
Tezimizde farklı makinelerin ayrımını sağlamak amacıyla etmen tabanlı gridin altyapısında kullanılan Jade mimarisinin sunduğu kapsayıcılar (container) kullanılmıştır. Grid sistemine dahil olacak her bir fiziksel makine için bir Jade kapsayıcısı yaratılarak sisteme eklenir. Böylelikle grid sistemi tarafından her bir kapsayıcıdaki işçilerin sisteme dahil farklı bir makinede çalışıyor oldukları bilinmektedir. Şekil 5.1 :‘de bir kapsayıcının yaratılması ve bu kapsayıcıya işçi etmen eklenmesi işlemine ilişkin kod parçası gösterilmiştir.

### 5.1.2 Bileşenlerin Ana Çatı ile İlişkileri

Kapsayıcılar yaratılırken Jade ana çatısının çalıştığı makinenin(sunucu) ip ve port bilgileri ile sisteme dahil edilirler. Tezimizde yönetici etmen Jade ana çatısının çalıştırıldığı makinede yaratılan bir kapsayıcı altında oluşturularak sisteme dahil edilmektedir. Aynı şekilde işçi etmenler de bulundukları makinede Jade ana çatısına bağlı bir kapsayıcı altında yaratılırlar.

```
//Kapsayıcı özellikleri belirleniyor, profil oluşturuluyor
ProfileImpl profile = new ProfileImpl(host, port, ID , false);
profile.setParameter(Profile.CONTAINER_NAME, "ContainerName");

//Oluşturulan profille kapsayıcı yaratılıyor.
AgentContainer container =
            Runtime.instance().createAgentContainer(profile);

//İşçi etmen yaratılarak kapsayıcıya ekleniyor.
WorkerAgent worker = new WorkerAgent();
worker.join(container, "WorkerName");
```

**Şekil 5.1 :** Kapsayıcı yaratılması ve bir işçi etmenin eklenmesi.




**Şekil 5.2 :** Kapsayıcı tabanlı sistem mimarisi

Tezimiz kapsamında bir kapsayıcı bir makineyi belirttiğinden, her makinede işçi etmenlerin bağlandığı sadece bir kapsayıcı bulunabilmektedir. Bu kısıtın sebebi göç hesaplamalarında farklı kapsayıcılarda çalışan işçilerin birbirlerinden farklı

makinelerde çalıştırıldığı kabulünün yapılmış olmasıdır. Böylelikle yük dengeleme için yapılacak göçlere karar verecek algoritma farklı taşıyıcılarda çalışan işçileri farklı makinelerde çalışıyor olarak ele alabilmektedir. İstemci etmenlerin bağlandığı kapsayıcılar için her makinede en fazla bir adet olma kısıtı bulunmamaktadır. Şekil 5.2 :'de tezimiz kapsamında Jade ana çatısının kullanımı ve kapsayıcı bazlı sistem mimarisi gösterilmiştir. Gösterilen mimaride yönetici etmen Jade ana çatısının bulunduğu makinede bulunmaktadır. Ancak Jade ana çatısının grid yapısına bağlı herhangi bir makinede çalışmasında herhangi bir kısıt bulunmamaktadır. Şekil 5.1 :'de gösterildiği gibi kapsayıcılar sunucu ip'si ve port numarası belirtilerek yaratılmaktadır. Seçilen ip ve port ana çatının çalıştığı makine ve ilgili portu gösterdiği sürece Jade ana çatısına istenilen makineden bağlanılabilir ve böylelikle ana çatı istenilen makinede çalıştırılabilir.

### 5.1.3 Görevlerin yapısı ve yürütülmesi

Tezimizde görev nesneleri etmen tabanlı grid sisteminden yapısal farklılıklar arz etmektedirler. Etmen tabanlı grid sisteminde yer alan görevler TaskRunner tarafından task.Task soyut sınıfına ait ve bu sınıftan türetilen görevlerde yer alan *execute* yordamına çağrı yapılarak çalışıtırılırlar. Bu görevler TaskRunner nesnesinin çalıştığı iş parçacığı(thread) altında çalışmaktadır.

Tezimizde ise task.Task soyut sınıfı java.lang.Runnable arayüzünden türetilmiştir ancak bu arayüzün gerektirdiği *run* yordamının gerçeklenmesi bu sınıftan türetilecek görev nesnelerine bırakılmıştır. Görevlerimiz tarafından yürütülen işler *run* yordamı içerisinde tanımlanmaktadır. Görevlerimizin java.lang.Runnable arayüzünden türetilmesiyle aslında görevlerin kendileri de birer iş parçacığı şeklinde işletilebilmektedir. Bu iş parçalarının yani görevlerimizin çalıştırılması Brakes kütüphanesinde yer alan be.ac.kuleuven.cs.ttm.computation.Computation sınıfı kullanılarak yapılmaktadır. task.Task yaratılırken sınıf üyesi olarak bir adet Computation nesnesi yaratılır. Computation nesnesi Brakes kütüphanesinde yer alan be.ac.kuleuven.cs.ttm.ttm.Factory sınıfı kullanılarak görevin kendisi de bu sınıfa ait *createComputation* yordamına parametre olarak verilerek yaratılır.Şekil 5.3 :'de Computation nesnesinin görevin yapıcısı içerisinde yaratılmasına ait kod parçası gösterilmektedir.

Computation sınıfı java.lang.Runnable arayüzünü gerçeklemiş bir iş parçacığını Brakes kütüphanesine ait be.ac.kuleuven.cs.ttm.computation. Scheduler isimli görev

sıralayıcı sınıfını kullanarak çalıştırmaktadır. TaskRunner sınıfı task.Task soyut sınıfına ait *startRunning* yordamını çağırarak görevin yürütülmesini sağlamaktadır. TaskRunner sınıfı TaskRunnerBase isimli bir arayüzden türetilmiştir. TaskRunnerBase arayüzü görevlere parametre olarak geçilerek görevler arası mesajlaşmaya ait metodlar görevler tarafından bu arayüz kullanılarak çağrılmaktadır. Böylelikle aynı arayüze sahip farklı özellikleri olan TaskRunner sınıfları türetilebilmesi esnekliği sağlanmıştır.

```
comp = Factory.getInstance().createComputation((Runnable)this);
```

**Şekil 5.3 :** Computation nesnesinin yaratılması.

Şekil 5.4 :'te *startRunning* yordamı gösterilmiştir. Şekilden görüldüğü üzere görevin göç ettirilme durumuna göre sınıf üyesi Computation(comp) nesnesine ait *resume* veya *start* yordamları çağrılır. Böylelikle görevimizin yürütülmesi olarak tanımlayabileceğimiz Computation göç durumuna göre görev sıralayıcı tarafından baştan başlatılarak veya kaldığı yerden devam ettirilerek yürütülür.

```
public void startRunning(TaskRunnerBase base) {
      tBase = base;
      if(isMigrated)
      {
            runningStarted(); //taskrunner bilgilendirilir
            m_IsCompleted = false;
            setWait(false); //brakes için gerekli
            comp.addListener(this);
            comp.resume(); //Görev devam ettirilir
      }
      else
      {
            runningStarted(); //taskrunner bilgilendirilir
            comp.addListener(this);
            comp.start(); //Görev başlatılır.
      }
}
```

**Şekil 5.4 :** Görev nesnesine ait *startRunning* yordamı.

Tezimiz kapsamında Brakes kütüphanesine ait Computation ve Scheduler sınıflarında değişiklikler yapılmıştır. Computation nesnesine görevleri yürütülme durumundan haberdar eden yapılar eklenmiştir. Scheduler sınıfına birden fazla iş parçacığının yani görevlerimizin güvenli şekilde çalışabilmesi(thread safety) için mekanizmalar eklenmiştir.

#### 5.1.3.1 Brakes Computation sınıfında yapılan değişiklikler

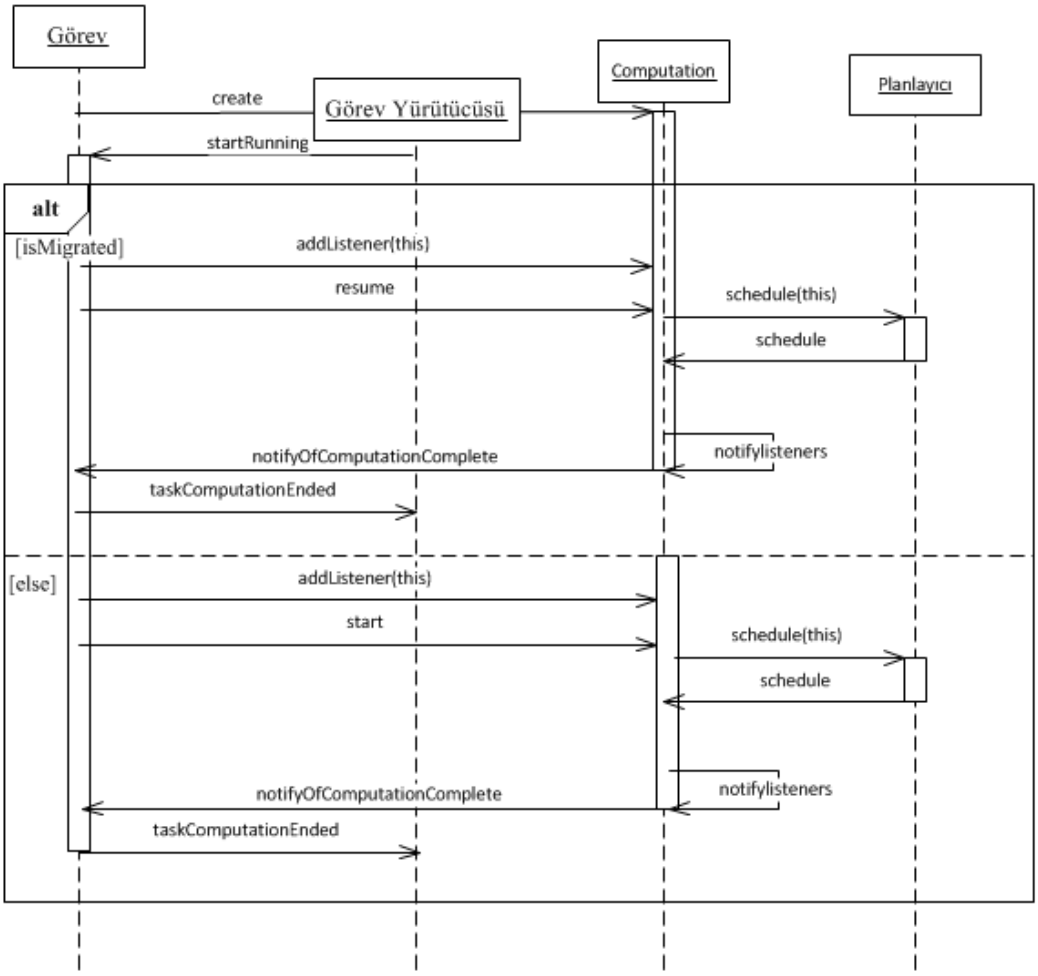
Brakes kütüphanesinde yer alan varsayılan Computation sınıfı gerçeklemesinde yürütülen görev tamamlandığında veya duraklatıldığında görev haberdar edilmez. Ancak tez çalışması kapsamında, bir görevin yürütülmesi sonlandığında sonuçların bildirilmesi veya göç emrini izleyen duraklatılma süreci tamamlandığında göç işlemine devam edilmesi için, yürütülen görevlerin Computation nesnesi tarafından işletimin tamamlandığı veya duraklatıldığından haberdar edilmesi gerekmektedir.

Bu amaçla, görev işletimi herhangi bir sebeple sonlandığında Computation sınıfına çalıştırdığı görevleri durumdan haberdar edecek bir yapı eklenmiştir. Şekil 5.4 :'de bu yapının bir parçası olan Computation nesnesine yapılan *addListener* yordam çağrısı görülmektedir. task.Task sınıfı task.ComputationListener arayüzünü gerçeklemektedir. Yine Computation sınıfı *addListener* yordamına task.ComputationListener arayüzüne sahip nesneleri kabul etmektedir. Bu şekilde Computation nesneleri görev yürütme işlemi sonlandığında veya duraklatıldığında ilgili görevleri *notifyListeners* yordamını yürüterek haberdar etmektedir. *notifyListeners* yordamı *addListener* yordamıyla Computation nesnesine eklenmiş görevlerin gerçeklediği ComputationListener arayüzüne ait *notifyOfComputationComplete* yordamını çağırır. İlgili yordamın task.Task sınıfında yapılan gerçeklemesinde görevler TaskRunnerBase arayüzünde tanımlanmış TaskRunner sınfında gerçeklenmiş *taskComputationEnded* yordamı çağrılır. Bu sayede TaskRunner tarafından sonuçların bildirilmesi veya göç başlatılması işlemleri gerçekleştirilebilmektedir. İlgili iş akışı Şekil 5.5 :'te tanımlanmıştır.

#### 5.1.3.2 Brakes Scheduler sınıfında yapılan değişiklikler

Brakes kütüphanesinde yer alan be.ac.kuleuven.cs.ttm.computation.Scheduler sınıfı iplikler(thread) açısından güvenli olarak tasarlanmamıştır. İşçi etmenlerin görevleri çalıştırırken kullandıkları TaskRunner sınıfı görevlerimizi içeren Computation nesnelerini çalıştırırken Scheduler sınıfını kullanmaktadır. Brakes tarafından Scheduler sınıfı static olarak tasarlanmıştır. Bu sebeple aynı kapsayıcıda yer alan işçilere ait tüm TaskRunner nesneleri static yapıdaki Scheduler.*getInstance()* yordamıyla edindikleri aynı Scheduler nesnesini kullanarak görev işletimlerini gerçekleştirirler. Bu sebeple iplikler açısından işletim güvenliği (thread safety) sağlamak amacıyla TaskRunner nesnelerinin görevleri çalıştırmak üzere çağırdıkları Scheduler.*getInstance()* yordamına çoklu erişimde hata çıkmasını engelleyecek bir

karşılıklı dışlamalı erişim yapısı eklenmiştir. Ayrıca Scheduler nesnesi içerisinde yer alan metodlara yürütülen Computation nesnelerinin Scheduler üzerinde çalıştırıldıkları iş parçacıklarının (thread) tutulduğu kuyruğa ve bu kuyruğa erişimde kullanılan indekslere güvenli erişime imkan sağlayacak karşılıklı dışlamalı erişim sağlayan mekanizmalar eklenmiştir. İlgili mekanizma gerçeklenirken Java diline ait **synchronized** anahtar kelimesi kullanılmıştır.




**Şekil 5.5 :** Computation sınıfına yapılan eklentilerin iş akışı.

### 5.1.4 Uygulanan Protokoller

Tezimiz kapsamında etmen tabanlı grid sistemine ait etmenlerin sisteme bağlanma protokolleri, görev atama protokolü, etmenlerin sistemden ayrılma protokolleri etmen tabanlı grid sisteminde olduğu şekliyle kullanılmıştır [1]. Tezimiz kapsamında mesajlaşma analizi ve görev göçü işlemlerini sağlamak için yeni protokoller tanımlanmıştır.

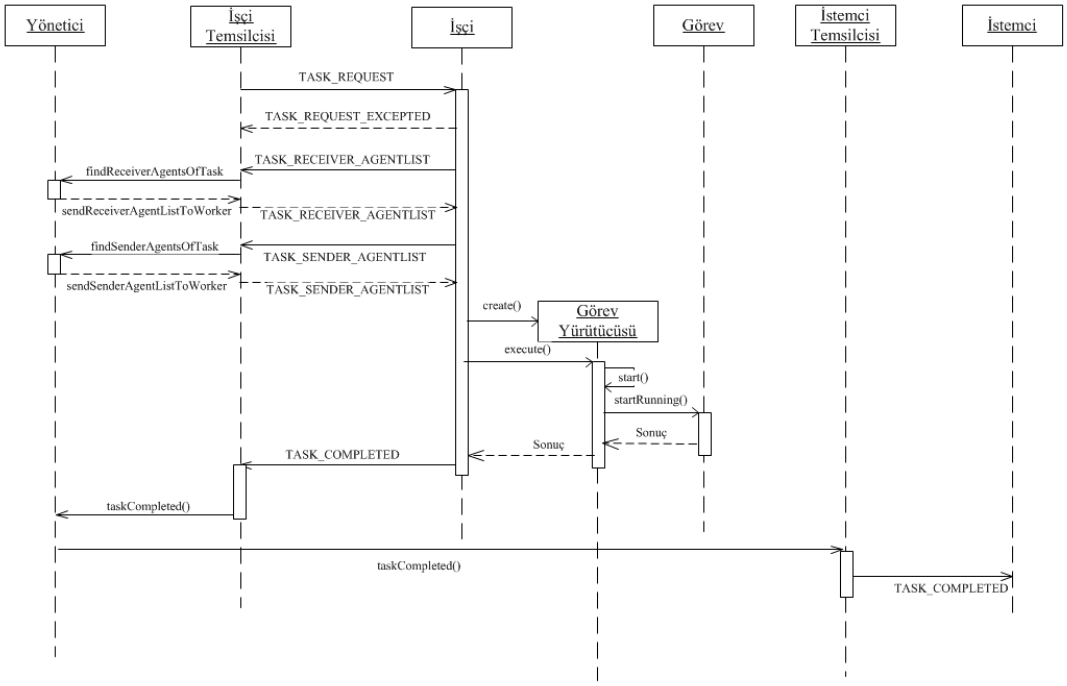
#### 5.1.4.1 Görev yürütülmesi ve sonuçlarının bildirilmesi

İşçi etmen, görev atama protokolüne göre yürütmesi gereken görevi aldıktan sonra bu işi yerine getirmek için çalışmaya başlar. Etmen tabanlı grid sisteminde bir işçi etmen bir anda sadece bir görevi çalıştırmakla yükümlüdür [1].

Görevin yürütülmesine dair protokol şu şekildedir.

1. İşçi temsilcisi görevi TASK_REQUEST mesajı ile işçi etmenine bildirir.
2. İşçi etmeni bu mesajı alınca TASK_REQUEST_ACCEPTED mesajını cevap olarak işçi temsilcisine yollar.
3. Yürütülecek görevin mesaj göndereceği hedef görevleri çalıştıran işçi etmenlerin listesi TASK_RECEIVER_AGENTLIST mesajı temsilci etmene gönderilerek yönetici etmenden talep edilir.
4. Yürütülecek görevin mesaj alacağı görevleri çalıştıran işçi etmenlerin listesi TASK_SENDERS_LIST_REQUEST mesajı temsilci etmene gönderilerek yönetici etmenden talep edilir.
5. Yönetici etmen ilgili görevin mesaj göndereceği görevlerin çalıştırıldıkları etmenlerin listesini TASK_RECEIVER_AGENTLIST mesajı ile, görevin mesaj alacağı görevlerin çalıştırıldıkları etmenlerin listesini TASK_SENDER_AGENTLIST mesajı ile işçi etmene gönderir.
6. Daha sonra işçi etmen, görevi yürütecek iş parçasını (Thread) oluşturacak olan worker.TaskRunner(görev yürütücüsü) sınıfından bir nesne yaratır.
7. TaskRunner nesnesi görevin execute yordamını yeni oluşturduğu bir iş parçasında çağırarak görevin işlevini yerine getirir.
8. Görevin yürütülmesi bitince TaskRunner nesnesi sonucu işçi etmene bildirir.
9. İşçi etmen sonucu TASK_COMPLETED mesajıyla işçi temsilcisine bildirir
10. İşçi temsilcisi görevin sonucunu yönetici etmene bildirir.
11. Yönetici etmen bu göreve ait istemci temsilcisini listesinden bulur ve görevin sonucunu istemci temsilcisine bildirir.
12. İstemci temsilcisi sonucu TASK_COMPLETED mesajıyla istemciye bildirir. İlgili görevi çalışan görevler listesinde çıkartır.

Şekil 5.6 :'da bu protokolün akışı görülmektedir.




**Şekil 5.6 :** Görev yürütülmesi ve sonuçların bildirilmesi.

### 5.1.4.2 Görevlerin durdurulması ve göç ettirilmesi

İşçi etmenler yürüttükleri görevleri, göç yönetici etmenin almış olduğu ve kendilerine mesajla bildirdikleri kararlar neticesinde durdurup, yeniden devam ettirilmek üzere diğer bir işçi etmene göç ettirebilmektedirler. Aynı şekilde görev göçünü kabul eden işçi etmen de taşınma işlemi tamamlanan görevi kaldığı yerden devam ettirebilir.

Bu işlemi gerçekleştiren protokol şu şekildedir:

1. Göç yönetici etmen tarafından göç analizi gerçekleştirilir; analiz sonuçlarına göre yönetici etmene hangi görevin hangi kapsayıcıdaki etmenlere göç ettirilmesi gerektiği bildirilir.

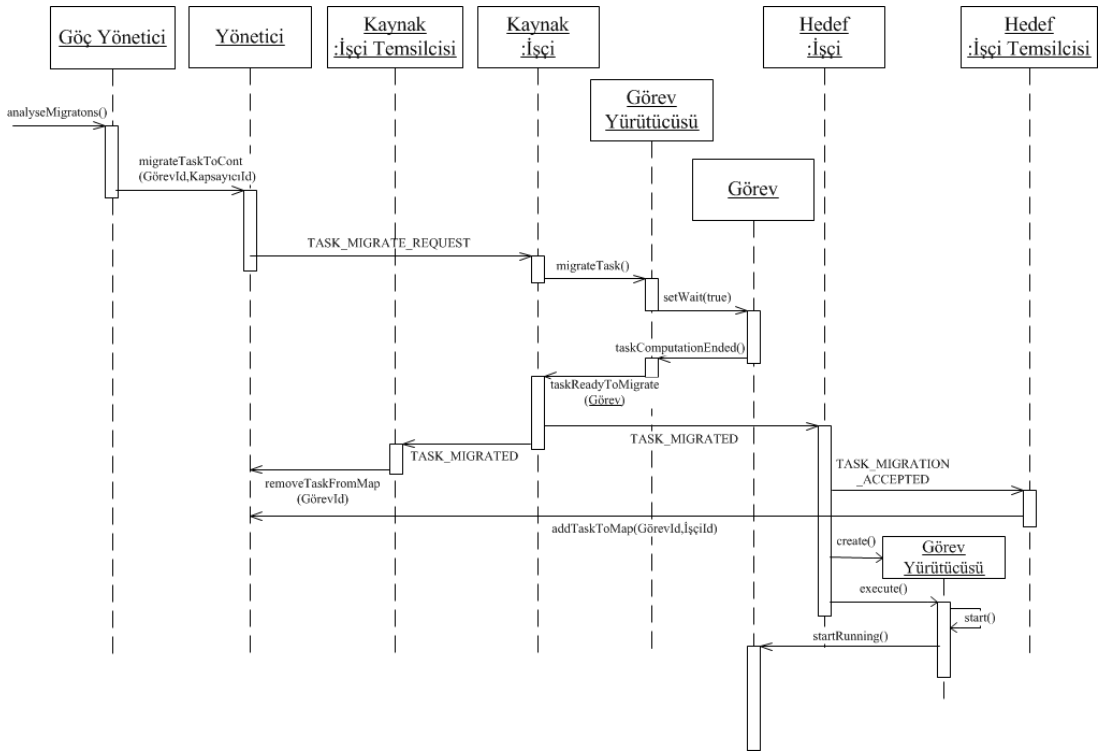
2. Yönetici etmen göç yöneticisi tarafından belirlenen kapsayıcıda göçü kabul edebilecek işçiyi belirler. Göç yöneticisinin karar algoritması bir görevi bir kapsayıcıya atamaya karar verdiyse, ya o kapsayıcıda görevi kabul edebilecek boşta bir işçi vardır ya da işçiler içerisinde çalıştırdığı görevi göç ettirecek en az bir işçi vardır. Boşta işçi olup olmamasına göre göç işlemi boşta işçi varsa hemen başlatılır, eğer boşta işçi yoksa hangi işçinin boşalacağı tespit edilir ve görev bu işçiye yönlendirilir.

3. Göç ettirilecek görevi çalıştıran işçiye TASK_MIGRATE_REQUEST mesajı

gönderilerek göç işlemi talep edilir. Görevin göç ettirileceği işçi etmenin sistemdeki id'si mesaja parametre olarak eklenir.

4. TASK_MIGRATE_REQUEST mesajını alan işçi etmen görev çalışmasını durdurur, görev kesintiye uğradığı anda sahip olduğu bağlam bilgisiyle birlikte serileştirilir. Görev sınıfında yer alan *isMigrated* ismindeki sınıf değişkeni "true" değerine kurulur.
5. Serileştirilen görev TASK_MIGRATED mesajına içerik nesnesi olarak eklenerek çalışmaya devam ettirileceği işçi etmene gönderilir.
6. Görev göçünü kabul eden işçi etmen görev kendisine ulaştığında temsilci etmenine TASK_MIGRATION_ACCEPTED mesajı gönderir. Göç ettirilen görev boşta olduğunda çalışmaya devam ettirilir. Eğer göçü kabul eden işçinin elinde göç ettireceği başka bir görev varsa önce göç işlemini tamamlar sonrasında yeni gelen görevi devam ettirir.

Şekil 5.7 :'de bu protokolün akışı görülmektedir.




**Şekil 5.7 :** Görev durdurulması ve göç ettirilmesi.

### 5.1.4.3 Görevlerin devam ettirilmesi

Görev göçünden sonra görevlerin devam ettirilmesi Bölüm 3'te ele alınan serileştirilmiş görevlerin içersindeki bağlam bilgilerinin kullanılmasıyla

gerçekleştirilir.

Bu işlemi gerçekleştiren protokol şu şekildedir:

1. İşçi etmen göç eden görevi yürütecek iş parçasını (Thread) oluşturacak olan worker.TaskRunner sınıfından bir nesne yaratır.
2. TaskRunner nesnesi görevin execute yordamını yeni oluşturduğu bir iş parçasında çağırarak görevi kaldığı yerden devam ettirir. Aslında görevin ilk kez çalıştırmasında ve devam ettirmesinde TaskRunner nesnesinin aynı yordamı(execute) çağrılır.
3. TaskRunner execute yordamın iş parçacığı altında görevi işletmeye başlar.
4. Görev nesnesi *isMigrated* ismindeki sınıf değişkeninin değerine bakar, eğer değişken kurulmuşsa önceki bağlam bilgisi kullanarak çalışmasına kaldığı noktadan devam eder. Eğer kurulmamışsa önceki bölümlerde anlatıldığı gibi görev baştan başlatılır.

Şekil 5.7 :'de bu protokolün akışı görülebilir.

#### 5.1.4.4 Görevler arası mesajlaşma protokolleri

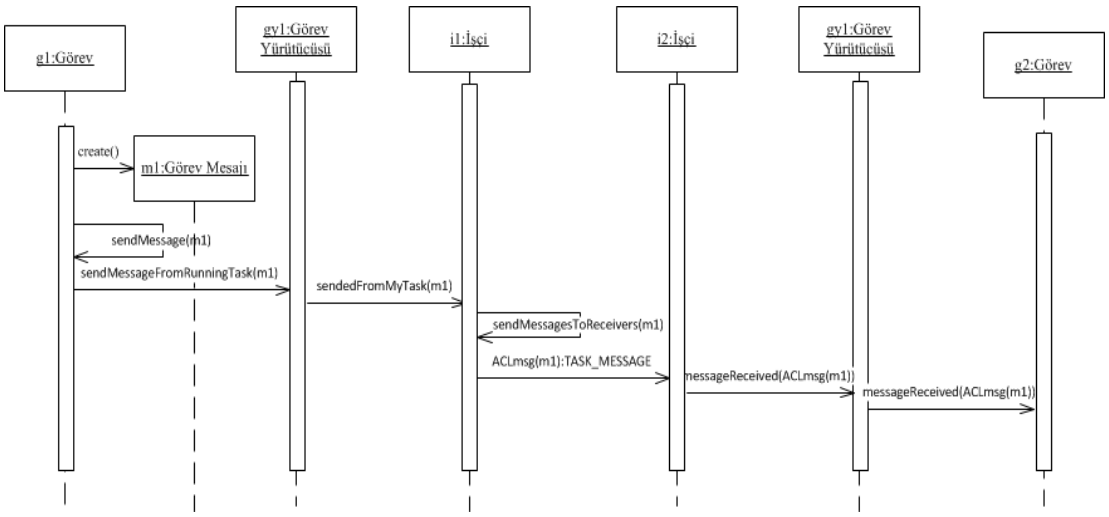
Tezimizde görevler kendi aralarında mesajlaşabilmektedir. Mesaj alma ve mesaj gönderme protokolleri şu şekilde işletilmektedir.

1. Görev işletimi devam ederken serileştirilebilir task.TaskMessage mesaj nesnesini oluşturulur. Görev task.TaskMessage nesnesi içeriğini gönderilmek istenen verilerle doldurur. task.TaskMessage sınıfının *sendMessage* yordamına ilgili mesaj iletilir.
2. *sendMessage* yordamı gönderilen mesajı task.TaskRunner sınıfına ait *sendMessageFromRunningTask* yordamına iletir.
3. TaskRunner nesnesi ise işçi etmene ait *sendedFromMyTask* yordamına çağrı yaparak mesajı işçi etmene iletir.
4. İşçi etmen TASK_MESSAGE tipinden bir ACL mesajı oluşturarak içerik nesnesi olarak serileştirilebilir TaskMessage nesnesini ekler. Mesaj dağıtımı multicast bir yapıdadır. Gönderilecek ACL mesajının alıcısı olan görevlerin çalıştığı işçi etmenlerin jade.core.AID tipinden sistem tanımlayıcıları, ACL mesajına ait *addReceiver(AID id)* yordamı kullanılarak mesaja eklenir.

Etmene ait send(ACLMessage) metodu ile oluşturulan ACL mesajı ilgili görevden mesaj alan görev veya görevlerin çalıştığı işçi etmen veya etmenlere gönderilir.

5. worker.WorkerAgentMsgHandler sınıfı içerisinde yer alan, işçi etmene gönderilmiş olan TASK_MESSAGE tipindeki mesajların alınmasını gerçekleştiren TaskCommunicationMessage sınıfından türetilmiş nesneye gönderilen TaskMessage tipindeki mesajlar ulaşır. Gelen mesajlar işçi etmenin *sendToMyTask* yordamı vasıtasıyla işçi etmene gönderilir.
6. *sendToMyTask* yordamı gelen mesajı task.TaskRunner sınıfına ait *messageReceived* yordamına iletir.
7. *messageReceived* yordamı gelen mesajı task.Task sınıfına ait *messageReceived* yordamına iletir ve böylelikle gelen mesaj işletilen görev tarafından alınmış olur.

Şekil 5.8 :'de bu protokolün akışı görülebilir.




**Şekil 5.8 :** Görevler arası mesajlaşma.

### 5.1.4.5 Görevler arası mesajlaşmaların bildirilmesi protokolü

Ağ trafik yükü dengeleme algoritması gerçekleştirilmesi gereken görev göçlerini belirlerken görevler arasındaki mesajlaşma sayılarını bir parametre olarak kullanmaktadır. Bunun sağlanması için mesajlaşma sayılarının göç yönetici etmene bir protokolle bildirilmesi gereklidir.

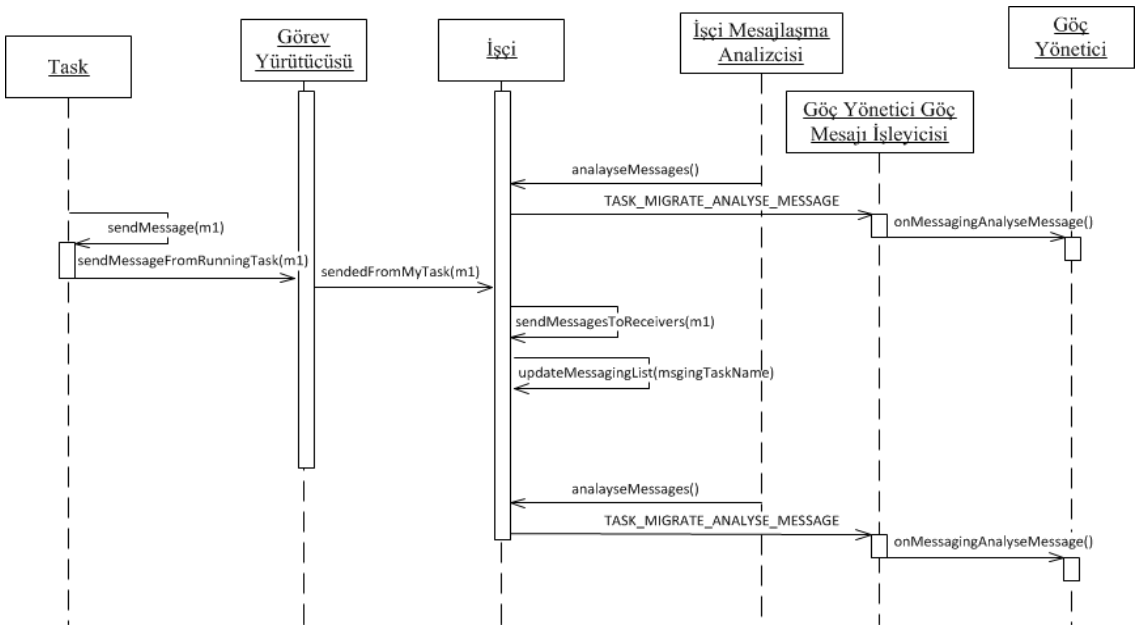
Bu işlemi gerçekleştiren protokol şu şekildedir.

1. İşçi etmen nesnesi görev işletimi sırasında diğer görevlere gönderilen ve

diğer görevlerden alınan tüm mesajları mesajlaşılan görevlerle eşleştirilerek bir listede tutar. Mesaj gönderilip alma işlemi sürerken liste de sürekli güncellenir. Örneğin A görevi B görevine x tane mesaj gönderip, y tane mesaj alırsa, A görevini çalıştıran işçi etmen mesaj sayıları listesinde B göreviyle $x + y$ adet mesajlaşma eşleştirir.

2. jade.core.behaviours.TickerBehaviour sınıfından türetilmiş worker.WorkerMessageAnalyser sınıfı belirlenen aralıklarla işçi etmende yer alan *analyseMessages* yordamına çağrı yapar.
3. *analyseMessages* yordamında TASK_MIGRATE_ANALYSE_MESSAGE mesajı oluşturularak içerik nesnesi olarak worker.TaskMessagingInfo nesnesi eklenir. TaskMessagingInfo nesnesi içeriğine görevin yapmış mesajlaşma listesinde yer alan mesajlaşma sayıları, görevin çalıştığı kapsayıcı bilgileri ve görevin gereksinim duyduğu özellikler eklenir. Oluşturulan mesaj göç yönetici etmene gönderilir.
4. TASK_MIGRATE_ANALYSE_MESSAGE mesajı migration.MigrationManagerMsgHandler tarafından alınarak göç yönetici etmene *onMessagingAnalyseMessage* yordamı kullanılarak iletilir.
5. *onMessagingAnalyseMessage* yordamı gelen mesajlaşma bilgilerini daha sonra yapılacak göç analiz hesaplamalarında kullanılmak üzere kaydeder.

Şekil 5.9 :'da bu protokolün akışı görülebilir.




**Şekil 5.9 :** Mesajlaşmaların bildirilmesi protokolü.

#### 5.1.4.6 Yük dengeleme algoritmasının çalıştırılması

Ağ trafik yükü dengeleme algoritması göç yönetici etmen tarafından belli aralıklarla çalıştırılır. Değişen görevler arası mesajlaşma sayıları ve sistemdeki etmenlerle ilgili değişiklikler göz önüne alınarak sistemde ağ trafiği dengeleme işlemi sürekli bir şekilde gerçekleştirilir. Bu sayede sistemdeki ağ üzerindeki trafik yükü durumu güncel olarak dağıtılmış olur.

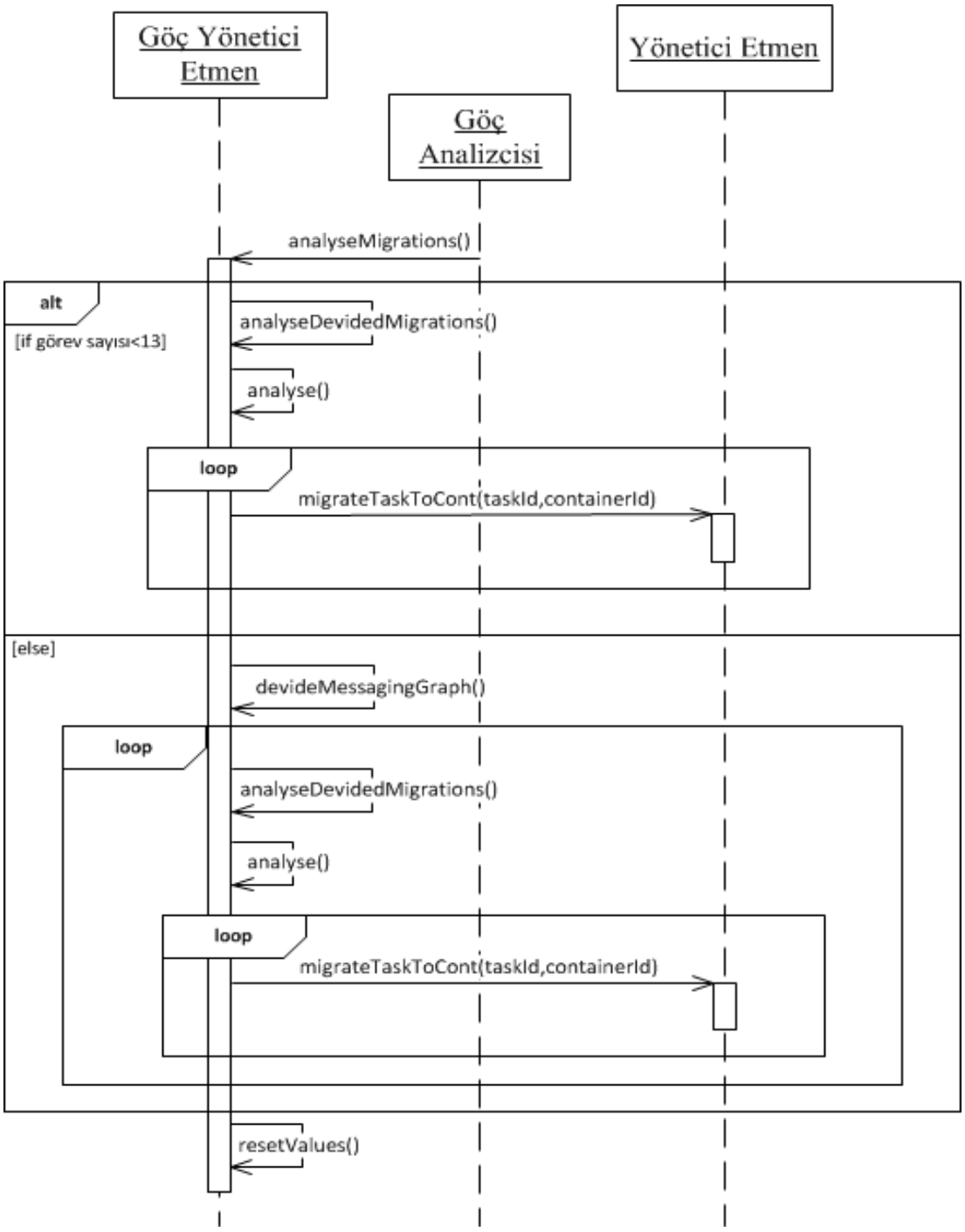
Bu işlemi gerçekleştiren protokol şu şekildedir.

1. MigrationAnalyser sınıfı JADE kütüphanesine ait TickerBehaviour sınıfından türetilmiştir. göç yönetici etmen TickerBehaviour sınıfının işleyişi gereği belirli sürelerle çağrılan *onTick* yordamında kendine ait *analyseMigrations* yordamına çağrı yapar. Bu çağrı ile ağ trafiği dengeleme için analiz işlemini başlatır.
2. Eğer sistemdeki toplam görev sayısı belirli bir sayının altında ise ağ trafik yükü analizi doğrudan başlatılır.
3. Eğer görev sayısı belirlenen değerden fazla ise görevler gruplara ayrılarak analiz yapılmaktadır. Bunun sebebi Jacop [5] kütüphanesinin bir ağaç taraması işlemi yaparak çözüm bulmasıdır. Görev sayısının lineer olarak artışı Jacop işlem süresini üstel olarak arttırmaktadır. Birbirleriyle mesajlaşan görevleri, mesajlaşılan görevlerin mesajlaştıkları görevleri özyinelemeli olarak bir gruba dahil ederek, göç etmeleri birbirlerini etkileyecek tüm görevler aynı gruba alınır. Görevleri düğümler, mesajlaşmaları ise kenarlar olarak düşünebileceğimiz bir graf yapısına dahil olacak tüm görevler bir gruba dahil edilerek analiz yapılır. Eğer oluşturulan alt gruplar yine beklenilenden yüksek sayıda görev barındırıyor iseler algoritmamız optimum çözümden vazgeçerek, istenilen sayıda görev barındıran alt gruplar oluşturmak için belli mesajlaşma sayısının altında mesajlaşan görevleri göz ardı ederek alt görev gruplarını oluşturulur. Daha sonra bu alt gruplar için analiz yapılır.
4. Kısıtlar tanımlanarak göçe karar verecek algoritma işletilir. Kısıtların analiz algoritmasına nasıl eklendiği ve bunun kararlar üzerindeki etkileri Bölüm 5.2'de örneklerle detaylı olarak ele alınacaktır. Analiz algoritması Bölüm 4'de ele alındığı şekliyle Jacop kütüphanesi kullanılarak

gerçeklenmiştir.

5. Yapılan analiz algoritmasının göç etmesine karar verdiği görevlerin çalıştıkları etmenlere göç kararları bildirilmek üzere ilgili göç kararları yönetici etmene iletilir.

Şekil 5.10 :'da bu protokolün akışı görülebilir.




**Şekil 5.10 :** Yük dengeleme algoritmasının çalıştırılması protokolü.

## 5.2 Sistem Testleri

Bu bölümde sistemimiz üzerinde gerçekleştirmiş olduğumuz bazı testlere yer verilecektir. Farklı kombinasyonlardaki etmen ve kapsayıcı sayısı, taşıma maliyetlerindeki ve mesajlaşmaların değişiminin algoritmanın verdiği kararlar üzerindeki değişimleri gözlemlenecektir. Algoritmanın sonucu olarak gerçekleşen göçü gösteren konsol çıktılarına yer verilecektir.

Test ortamımızda yer alan etmenlerimiz işçi etmenleri, kapsayıcılarımız grid üzerinde konulanmış makineleri göstermektedir. Taşıma maliyeti ise bir görevin bir kapsayıcıdan yani bir makineden diğer bir kapsayıcıya yani makineye taşınması sırasında sisteme getirdiği yükü belirtmektedir.

Kapsayıcı, etmen ve bir görevin göç maliyeti gibi parametreler her test ortamı için aşağıdaki şekilde ifade edilecektir:

Sistem için görev göçü maliyeti: X

Kapsayıcı n:

- x tane işçi etmen.
- Sağladığı özellikler(donanım vs.): K, L (özellik tanımlayıcıları)

Görevler ve talep ettikleri özellikler:

Görev(n) : M

Görev(n+1) : Yok

### 5.2.1 Sistem Testi 1

Bu testimizde iki kapsayıcı bulunan bir sistemde yürütülen dört görev yer almaktadır.

Sistem Test 1 Ortamı:

Sistem için görev göçü maliyeti: 19

Kapsayıcı(1):

- 2 tane işçi etmen.
- Sağladığı özellikler: B

Kapsayıcı(2):

- 3 tane işçi etmen.

- Sağladığı özellikler: A

Görevler ve talep ettikleri özellikler:

Görev(1): A

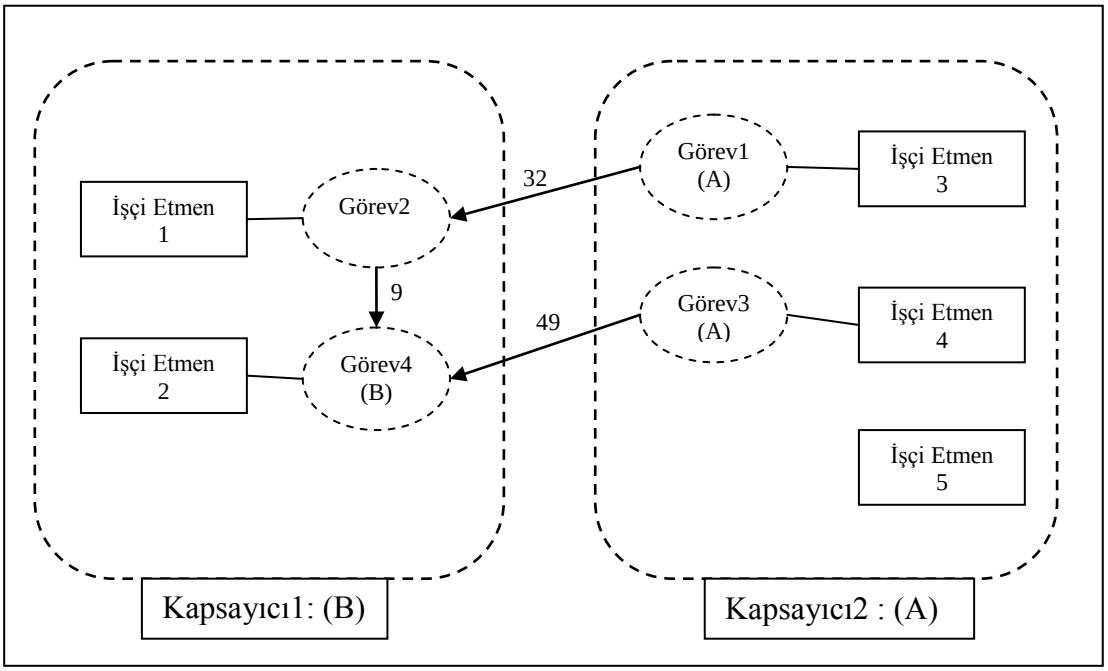
Görev(2): Yok

Görev(3): A

Görev(4): B

Etmen tabanlı grid sistemi talep ettikleri özelliklere göre görev(1)'i ve görev(3)'ü kapsayıcı(2)'ye, görev(4)'ü ise kapsayıcı(1)'deki uygun bir işçi etmene atanır. Görev(2) ise talep ettiği bir özellik olmadığından rastgele olarak boştaki bir kapsayıcıya atanır. Sistem bir süre işletildikten sonra ağ trafiği dengeleme algoritması işletilir.

Şekil 5.11 :'de ağ trafiği dengeleme algoritması işletilmeden önce görevlerin çalıştırıldıkları yerler ve bu ana kadarki mesajlaşmaları görülmektedir. Yönlü oklar görevler arası mesajlaşmaların yönünü okları üzerindeki sayılar mesajlaşma sayılarını belirtmektedir.




**Şekil 5.11 :** Test1 ortamında ağ trafiğindeki dengeleme öncesi durum

Ağ trafiği dengeleme algoritmasının yapısı ve kısıtların tanımlanması Bölüm 4'te ele alınmıştır. Bu bölümde ağ trafik yükü dengeleme algoritmamızın daha iyi anlaşılması açısından, test1 ortamının algoritmamıza tanıtılması ve yapılan analiz işlemi aşağıda

maddeler halinde gösterilmektedir.

- Değişkenler kümesi görevlerimizden oluşmaktadır. Sistem Test1 ortamındaki her bir görev için [1-2] arasında değer alabilen değişkenler tanımlanmıştır. İlgili değerler sistemde var olan kapsayıcıların tanımlayıcılarıdır(id). Sistemdeki her bir kapsayıcının nümerik bir tanımlayıcısı bulunmaktadır. Görev değişkenlerine bu değerlerden atanması görevin o kapsayıcıda çalışacağı anlamına gelmektedir.
- Kapsayıcılar arası yapılan mesajlaşmalar algoritmamıza maliyet olarak eklenir. Örneğin görev(1) ile görev(2) arası mesajlaşma eğer aynı kapsayıcıda yer alırlarsa maliyetsiz, ancak farklı kapsayıcıda yer alırlarsa mesaj sayısı kadar yani 32 maliyeti olacak şekilde koşullu olarak algoritmaya eklenir. Tüm görevler arası mesajlaşmalar için aynı işlemler yapılır.
- Görevlerin taşınmasının sisteme getireceği maliyetler algoritmamıza kısıt olarak eklenir. Bu işlem şu şekilde yapılır. Yük dengeleme algoritması değişkenlerin öncül değerlerini yani görevlerin algoritma işletilmeden önceki yerlerini bilmektedir. Algoritmanın işletilmesi sonucunda gerçekleşecek değer atamalarında öncül değerlerinden farklı bir değer ataması yapılacak değişkenler için toplam maliyetin 19 arttırılması kısıtı eklenir. Yani ağ trafiği dengeleme algoritmamıza bir nevi "eğer üreteceğin sonuçta bir görevi başka bir kapsayıcıya(makineye) taşıyacaksan maliyeti 19 arttırman gerekir, gerekli hesaplamaları bu maliyet artışını gözeterek yapmalısın" denilir. Örneğin görev(3)'ün değerinin 1 iken 2 olması durumu maliyeti 19 arttırır. Algoritma hesaplamalarını bunu göz önünde bulundurarak gerçekleştirir.
- Görevlerin talep ettikleri özellikleri barındıran kapsayıcılarda çalışması gerektiği kısıtı algoritmaya eklenir. A özelliğini talep eden görev(1) ve görev(3) görevleri kapsayıcı2'de veya sistemde yer alan bu özelliği sağlayacak bir kapsayıcıda, B özelliği talep eden görev(4) ise kapsayıcı1'de veya sistemde yer alan bu özelliği sağlayacak bir kapsayıcıda çalıştırılması kısıt olarak eklenir. Yani bu örnek için ele alacak olursak Jacop değişkenleri olarak düşünürsek atama yapılacak değişkenlerden görev(1) ve görev(3)'e ait olanlarına 2, görev(4)'ü temsil eden değişkene bir ataması yapılmış olunur.
- Kapsayıcılarda yer alan işçi sayıları göz önünde bulundurularak kapsayıcıda

çalışabilecek en az ve en fazla sayıdaki görev sayıları kısıt olarak algoritmaya eklenir. İşçi etmen sayıları göz önünde bulundurulduğunda, kapsayıcı1 için görev sayısı aralığı [0-2] iken kapsayıcı2 için [0-3]'tür. Algoritmamız kapsayıcı1 için en fazla iki, kapsayıcı2 için en fazla üç görev ataması yapacak şekilde kısıtlanmış olur.

- Jacop[5] kütüphanesi bize kısıtlarla programlama yapmayı sağlar. Tanımlanan kısıtlar bir problemin çözümünde kullanılacak bir algoritmayı oluşturur. Jacop tarafından belirlenen bir değişkenin(kısıt değişkeni) değerini minimize edecek sonucu bulacak şekilde çözüm aramaları yapılabilmektedir. Mesajlaşma ve taşıma maliyetleri toplamının bir değişkene atanması ve bu değişkenin minimize edecek çözümün aranması Jacop[5] tarafından bize sunulan ve kısıt çözüm algoritmamızda kullandığımız bir özelliktir. Bu özellik sayesinde problemimiz en düşük maliyetle çözülmeye kısıtlanmış olur.

Ağ trafik yükü dengeleme algoritmamız yukarıda tanımladığımız tüm kısıtları karşılayacak sonuçların bulunmasıyla işletilmiş olur. Sistem Test1 ortamını özelinde ele alırsak algoritmamız şu şekilde çalışır:

Öncelikle belirli özellikleri talep eden görevlerin değişkenlerine bu özellikleri sunan kapsayıcılara atanır. Yani görev(4) = 1, görev(1)=2, görev(3)=2 olur. Bu haliyle görev(4) ile görev(3) arasında gerçekleşen 49 mesajlaşma farklı kapsayıcılarda olacakları kesin olduğundan algoritma tarafından maliyet olarak eklenir. Görev(4) görev(1) ve görev(3) ilk çalıştıkları kapsayıcılarda kaldıklarından taşıma maliyetleri yoktur. görev(2) herhangi bir özellik kısıtı bulunmadığı için her iki kapsayıcıda yer alabilir. Algoritma kısıtlar dahilindeki tüm atamaları deneyerek en düşük maliyetli çözümü arar.

Görev(2)=1 ataması yapıldığı durumda görev(2) ile görev(1) arasındaki 32 mesajlaşma farklı kapsayıcılar arası olacağından maliyet olarak eklenir, görev(2) ilk durumda da kapsayıcı1'de bulunduğundan görev(2)=1 ataması için taşınma maliyeti yoktur. Bu durumdaki toplam maliyet $49 + 32 = 81$ olur.

Görev(2) = 2 ataması yapıldığı durumda görev(2) kapsayıcı1'den kapsayıcı2'ye taşınmış olacağından taşınma maliyeti(19) toplam maliyete eklenir. Görev(2) ile görev(4) arasında gerçekleşen 9 mesajlaşma taşınmadan sonra farklı kapsayıcılar

arası olacağından maliyet olarak eklenir. Toplam maliyet $49 + 19 + 9 = 77$ olur.

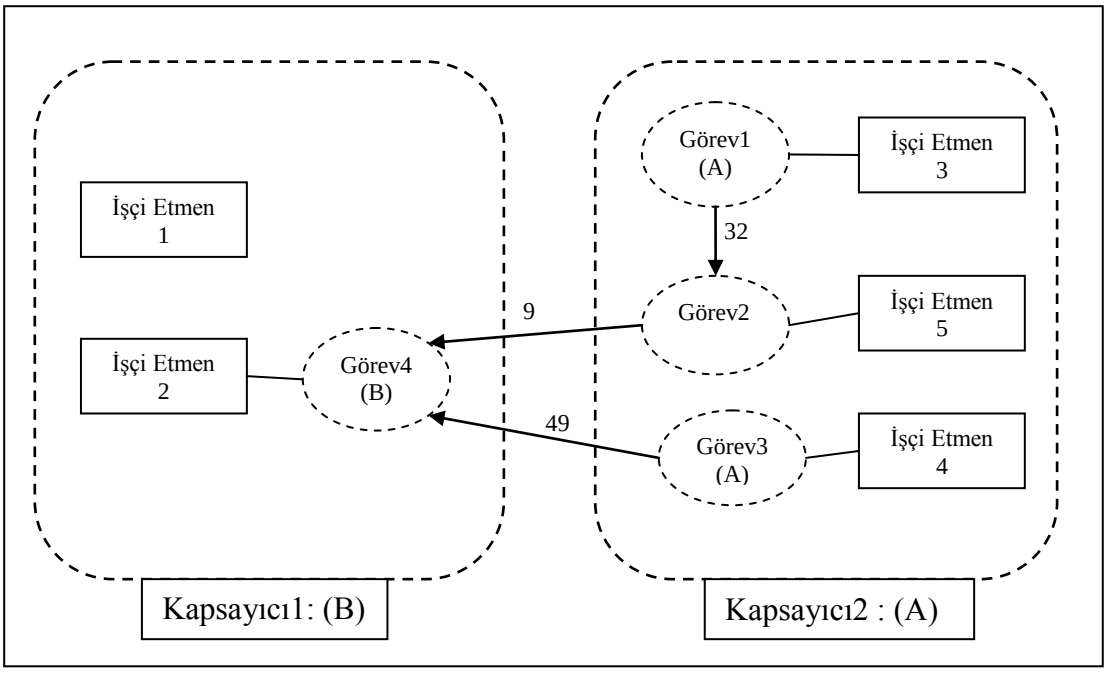
Algoritma 77<81 olduğundan görev(2)'nin taşınmasına karar verir. Mesajlaşma sayıları aynı şekilde devam ettiği takdirde sistem test1 ortamı Şekil 5.12 :' daki gibi olur. Aşağıda algoritmanın işletilmesi sırasında oluşan, göç işleminden önce ve sonra gerçekleşen analizlere ait konsol çıktısı görülmektedir:

```
**** Mesaj Analizi yapiliyor***** //Sistem Test1 ortamı analizi
MATRIS TABLOSU //Mesajlaşma matrisi
0  49  9  0
49  0  0  0
9  0  0  32
0  0  32  0

Initial positions: { //İlk değerler
CLIENTTestPro0@zaff:1099/JADE3 cont: 1 //görev(4)
CLIENTTestPro0@zaff:1099/JADE2 cont: 2 //görev(3)
CLIENTTestPro0@zaff:1099/JADE1 cont: 1 //görev(2)
CLIENTTestPro0@zaff:1099/JADE0 cont: 2 //görev(1)
}

Solution cost is 77
Depth First Search DFS0
[Task1 = 1, Task2 = 2, Task4 = 2, Task3 = 2]
Solution : [Task1=1, Task2=2, Task3=2, Task4=2]
Cost 77
Nodes : 2
Decisions : 1
Wrong Decisions : 1
Backtracks : 1
Max Depth : 1

CLIENTTestPro0@zaff:1099/JADE1
Göç analizi basariyla tamamlandi.
*********Mesaj Analizi bitiyor***********
Computation.block //Göç gerçekleştiriliyor.
notified
message sended to:( agent-identifier :name Worker20@zaff:1099/JADE
:addresses (sequence http://192.168.1.90:7778/acc )) //Göç mesajı.
**** Mesaj Analizi yapiliyor***** //Bir sonraki analiz
MATRIS TABLOSU
0  49  5  0
49  0  0  0
5  0  0  32
0  0  32  0

Initial positions: {
CLIENTTestPro0@zaff:1099/JADE3 cont: 1
CLIENTTestPro0@zaff:1099/JADE2 cont: 2
CLIENTTestPro0@zaff:1099/JADE1 cont: 2
CLIENTTestPro0@zaff:1099/JADE0 cont: 2
}

Solution cost is 54
Depth First Search DFS1
[Task1 = 1, Task2 = 2, Task4 = 2, Task3 = 2]
```

```
Solution : [Task1=1, Task2=2, Task3=2, Task4=2]
Cost 54
Nodes : 2
Decisions : 1
Wrong Decisions : 1
Backtracks : 1
Max Depth : 1

Göç analizi basariyla tamamlandı.
*********Mesaj Analizi bitiyor***********
```

Yukarıdaki konsol çıktısından görüldüğü üzere taşıma işlemi sonrasında gerçekleşen mesajlaşmalar sonrasında ağ trafiği dengeleme algoritmasının yeniden işletilmesiyle göç kararı verilmemektedir ve çözüm maliyeti 54 olmaktadır ki bu da yukarıda mesajlaşma matrisinde görülen farklı kapsayıcılar arası mesajlaşmaların toplamına yani 49+5'e eşittir.

Algoritma işletilmeden önce 81 olan kapsayıcılar arasındaki mesajlaşma sayıları, hesaplamalardan sonra 58 olarak hesaplanarak karar verilmiştir. Görev göçlerinden sonra kapsayıcılar arası mesajlaşma %28,4'lük bir azalma göstermiştir.




**Şekil 5.12 :** Sistem Testi 1 ortamında ağ trafiği dengeleme sonrası durum

### 5.2.2 Sistem Testi 2

Bu testimizde daha 6 makine ve 40 işçinin yer aldığı ve daha çok mesajlaşmanın olduğu bir sistemi ele alacağız. Aşağıda sistemde yer alan kapsayıcılar ve görevlerle ilgili bilgiler verilmiştir.

Sistem Testi 2 Ortamı:

Sistem için görev göçü maliyeti : 19

Kapsayıcı(1):

- 6 tane işçi etmen.
- Sağladığı özellikler: A
- Görev(1): A, Görev(2): A, Görev(3): Yok, Görev(4): Yok

Kapsayıcı(2):

- 7 tane işçi etmen.
- Sağladığı özellikler: B
- Görev(5): B, Görev(6): B, Görev(7): Yok, Görev(8): Yok

Kapsayıcı(3):

- 6 tane işçi etmen.
- Sağladığı özellikler: B,C
- Görev(9): B, Görev(10): C, Görev(11): Yok, Görev(12): Yok

Kapsayıcı(4):

- 7 tane işçi etmen.
- Sağladığı özellikler: A,D
- Görev(13): D, Görev(14): A, Görev(15): Yok, Görev(16): Yok

Kapsayıcı(5):

- 7 tane işçi etmen.
- Sağladığı özellikler: E
- Görev(17): E, Görev(18): E, Görev(19): Yok, Görev(20): Yok

Kapsayıcı(6):

- 7 tane işçi etmen.
- Sağladığı özellikler: E,F
- Görev(21): E, Görev(22): E, Görev(23): Yok, Görev(24): Yok

Görevler arası mesajlaşmalar matris olarak Şekil 5.13 :'de verilmiştir.

Yapılan analiz işlemi ve uygulanan göçler sonrasında 512 olan kapsayıcılar arası mesajlaşma sayısı 217'ye düşürülmüştür.

| Görev Id | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 8 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 50 | 0 |
| 3 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 55 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 29 | 0 | 0 | 0 | 0 | 0 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 60 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 29 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 21 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 25 | 50 | 0 | 25 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 40 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 20 | 0 | 0 |
| 13 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 70 | 0 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 30 | 0 | 0 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 70 | 0 | 0 | 0 | 0 | 44 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 50 | 0 | 0 | 0 | 0 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 18 | 0 | 0 | 55 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 44 | 0 | 0 | 0 | 0 | 44 | 0 | 0 | 0 | 0 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 25 | 0 | 0 | 0 | 0 | 0 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 0 | 0 | 0 | 0 | 60 | 0 | 0 | 0 | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 44 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | 0 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | 0 | 0 | 0 | 0 | 0 | 0 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Şekil 5.13 :** Sistem Testi 2 ortamı mesajlaşma matrisi.

### 5.3 Ağ Trafiği Dengeleme Algoritması Sınama Ortamı

Tezimiz kapsamında bilgisayar biliminde farklı alanlardan birçok öğeyi birleştirerek kullanmaktayız. Jade tabanlı bir etmen tabanlı grid sistemi üzerinde, Brakes kütüphanesi kullanarak kuvvetli görev göçü desteği sağlamaktayız. Ağ trafik yükü dengeleme algoritması ise Jacop [5] kütüphanesi kullanarak geliştirildi.

Testler sırasında asıl ilgi alanımız yük dengelemelerde kullanılacak karar mekanizmasının kısıt karşılama metodolojisine uygun hale getirilip gerçeklenmesinin sağlanmasıdır. Görevlerin hangi ortamda, nasıl çalıştığı veya nasıl göç ettiği dengeleme algoritması açısından bir önem arz etmemektedir. Dolayısıyla göçe karar veren algoritmanın diğer bileşenlerden ayrılarak, algoritma tarafından verilen kararların doğruluğunun harici olarak test edilebilmesinin sağlanması hem ağ trafiği dengeleme algoritmasının iyi yönde geliştirilmesi, hem de gerçek sistemin çalıştırılırken veya ilklendirilirken oluşturabileceği zaman kaybının azaltılması açısından önemlidir. Bu sebeple tezimiz kapsamında etmen tabanlı grid sistemini ve göç etme mekanizmasını emüle ederek ağ trafik yükü dengeleme algoritmasına girdi sağlayan bir simülasyon ortamı geliştirilmiştir.

Sınama ortamımızdaki ağ trafiği dengeleme algoritması gerçek sistemdeki algoritmanın bir farkla aynısıdır denilebilir. Gerçek sistemde gerçek bir grid üzerinde çalışan gerçek görevlerin gerçek mesajlaşma bilgileri kullanılarak karar verilmektedir, ancak sınama ortamında grid, görevler ve mesajlaşmalar ile ilgili gerekli tüm veriler arayüzler yardımıyla belirlenmekte veya belirlenen katsayılara göre otomatik olarak üretilmekte, bu bilgiler kullanılarak analiz yapılmaktadır. Sınama ortamı bize grid, görevler ve mesajlaşmalarla ilgili istediğimiz durumu ve ortamı oluşturarak, oluşturduğumuz sanal ortamda ağ trafik yükü dengeleme algoritmamızın vereceği kararları ve sonuçlarını gözlemleme imkanı sunmaktadır.

### 5.3.1 Sınama Aracının Kullanılması

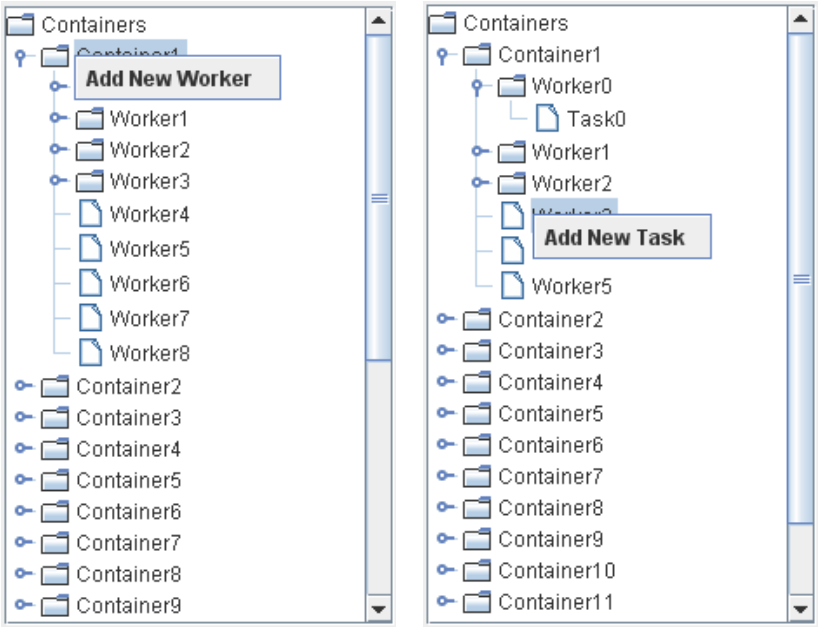
Sınama aracımızda grid öğeleri, görevler ve mesajlaşmalardan oluşan sanal bir etmen tabanlı grid sistemi yaratılabilmektedir. Şekil 5.15 :'te görülen sınama aracı arayüzünde oluşturulan sanal sistemde yer alan görevler ve kapsayıcılar listeler halinde gösterilmektedir. Ayrıca bu arayüz kullanılarak yeni kapsayıcılar tanımlanabilmektedir. Tanımlanan kapsayıcılara işçi etmenler atanabilmekte ve işçi etmenlere görevler eklenebilmektedir. Girilen kapsayıcı ve işçi sayılarına göre rastgele test ortamları oluşturulabilmektedir bu sayede farklı ortamlarda algoritmanın test edilmesi mümkün olabilmektedir. Görevlerin bağımlılıkları ve diğer görevlerle mesajlaşmaları ile ilgili veriler Şekil 5.16 :'da görülen görev özellikleri arayüzünden ayarlanabilmektedir.

"Add New Container" düğmesine basılarak simülasyon ortamına yeni bir kapsayıcı eklenir. Eklenen kapsayıcı kapsayıcılar listesine eklenir. Şekil 5.14 :'de gösterildiği gibi kapsayıcılar listesinde herhangi bir kapsayıcıya sağ tıklanıp "Add New Worker" sekmesi seçilerek ilgili kapsayıcıya yeni işçiler eklenebilmektedir.

"Add New Task" düğmesine basılarak simülasyon ortamına yeni bir görev eklenir. "Add New Task" düğmesine basıldığında eğer "Edit While Creating" seçeneği aktifse Şekil 5.16 :'da yer alan görev özellikleri arayüzü otomatik olarak açılır. Kapsayıcılar listesinde eklenen işçilere sağ tıklanarak Şekil 5.14 :'de gösterilen "Add New Task" sekmesi açılır. Bu sekme ile seçilen işçiye yeni görev ataması yapılabilmektedir.

Görev özellikleri arayüzünden sanal görevlerle ilgili çeşitli özellikler ayarlanabilmektedir. "Actual Worker" kısmından görevin çalışacağı işçi etmen

seçilebilmektedir. “Messaging” kısmından görevin mesajlaşacağı görev seçilip, mesajlaşma sayısı girildikten sonra “Add/Change” düğmesine basılarak görevler arası mesajlaşmalar tanımlanır. “Property” kısmında ise görevin talep ettiği özellik tanımlanabilmektedir.

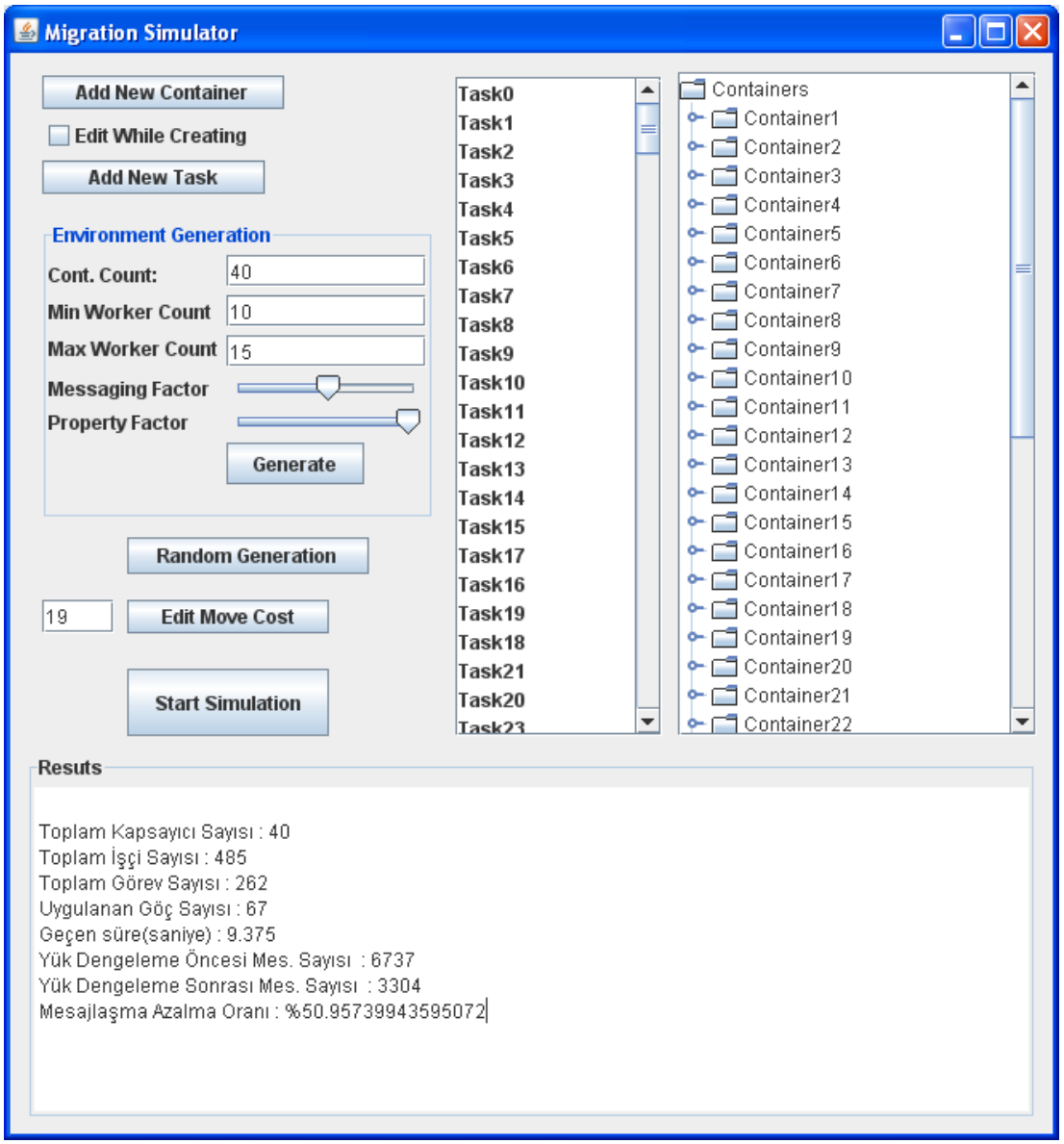



**Şekil 5.14 :** Kapsayıcı ve işçilere sağ tıklandığında açılan menüler.

Sınama aracı yardımıyla simülasyon ortamı oluşturulurken tek tek kapsayıcı ve işçilerin oluşturulması zaman aldığından hızlı bir şekilde test ortamları oluşturma seçenekleri sınama aracına eklenmiştir. Sistemdeki kapsayıcı sayısı ve her bir kapsayıcıda yer alabilecek minimum ve maksimum sayıdaki işçi etmen sayıları belirlenip, mesajlaşma ve görevlere ait özellik barındırma katsayıları kaydıraçlar yardımıyla seçilip “Generate” düğmesine basılarak ilgili özellikte bir test ortamı rastgele şekilde oluşturulabilmektedir. Kapsayıcılara barındırdıkları işçi sayıları ve sağladıkları özellikler göz önüne alınıp rastgele sayılarda görevler atanmakta ve bu görevler arasında seçilen katsayıya uygun olarak rastgele sayılarda mesajlaşmalar belirlenmektedir. “Random Generation” düğmesine basılarak, rastgele sayılarda kapsayıcı, işçi, görev ve görevler arası mesajlaşmaların bulunduğu bir sistem otomatik olarak oluşturulabilmektedir.

Etmen tabanlı grid ortamı elle veya rastgele bir şekilde sanal olarak oluşturulduktan sonra “Start Simulation” düğmesine basılarak ağ trafiği dengeleme algoritması çalıştırılmaktadır. Bu işlemden sonra alınan kararlar doğrudan sanal etmen sisteminde işletilmektedir. Farklı kapsayıcılarla(makinelerle) olan mesajlaşma

sayılarının ağ trafiği dengeleme işleminden önceki ve sonraki halleri görüntülenerek, algoritmanın ağ yükünü ne ölçüde etkilediği gözlemlenebilmektedir.



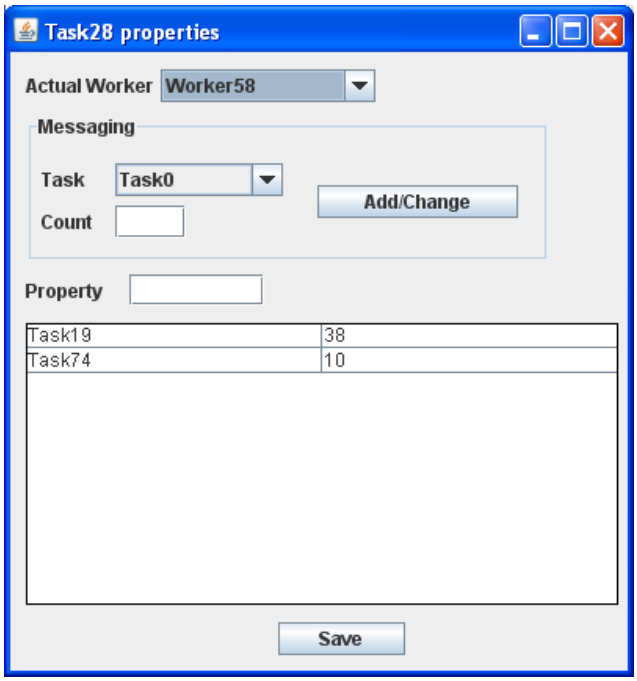

**Şekil 5.15 :** Sınama aracı arayüzü.

### 5.3.2 Testler ve Sonuçları

Sınama aracımızda çeşitli sanal ortamlar yaratarak algoritmamızın bu ortamlarda verdiği kararlar uygulandığında farklı kapsayıcılar arasında gerçekleştirilen mesaj sayılarında azalmalar görünmektedir.

Sınama aracı ara yüzünde kapsayıcılar arasında gerçekleşen ağ trafik yükü dengeleme algoritması çalıştırılmadan önceki ve sonraki mesajlaşmaların sayıları gözlemlenebilmektedir. Bu bölümde çeşitli sayılarda kapsayıcı ve işçi barındıran rastgele şekilde üretilmiş simülasyon ortamlarında algoritmamızın işletimi

sonucunda karar verilen göç sayıları, kapsayıcılar arasındaki mesajlaşmaların ne yönde etkilendiği gözlemlenecektir.




**Şekil 5.16 :** Görev özellikleri arayüzü.

Her bir test ortamında gerçekleştirilen denemeler için mesajlaşma karmaşıklıklarının ve görevlerin özellik barındırma yoğunluklarının rastgele olarak belirlenmesinde kullanılan katsayılar belirlenmiştir. Mesajlaşma ve özellik katsayıları adını verdiğimiz bu katsayılar [0-15] arası değerler alabilmektedir. Mesajlaşma katsayısı 15 belirlendiğinde görevler arasında yüksek yoğunlukta mesajlaşmalar rastgele şekilde üretilir, benzer şekilde 0 değeri verildiğinde mesajlaşma yoğunlukları düşük olacak şekilde görevler arası mesajlaşmalar belirlenir. Özellik katsayısı 15 verildiğinde görevler yoğunluklu olarak bir özellik talep edecek şekilde rastgele üretilir, 0 verildiğinde ise özellik talep etmeyen yani bir bağımlılığı bulunmayan görevlerin üretilme yoğunluğu artar. Farklı rastgelelik parametreleri kullanılarak farklı özelliklerde ortamların oluşturulması ile algoritmamızın farklı özelliklerdeki ortamlardaki davranışlarının incelenmesi amaçlanmıştır.

#### 5.3.2.1 Test1 ortamı

Test1 ortamında kapsayıcı sayısı 40, kapsayıcı başına düşen işçi sayısı 8, görevlere ait taşınma maliyetinin 19 ve kapsayıcı başına düşen görev sayısının 4 olduğu bir sistem ele alınmıştır. Test1 ortamına ait farklı mesajlaşma ve özellik katsayılarında yapılan testlerin sonuçları aşağıda verilmiştir.

Oluşturulan tüm test ortamlarına ait test durumları Çizelge 5.1 :'de gösterildiği gibi tablolar halinde ifade edilmiştir. Bu tablolarda yer alan "Dengeleme Öncesi Mesaj Sayısı" ve "Dengeleme Sonrası Mesaj Sayısı" tablo alanları, ağ trafiği dengeleme öncesinde ve ağ trafiği dengelendikten sonra tüm sistemde geçekleşen kapsayıcılar arası mesajlaşmaların sayılarını belirtmektedir. Benzer şekilde "Mesajlaşma Azalma Oranı" kapsayıcılar arası mesajlaşmalardaki azalma oranını belirtmektedir."Özellik Katsayısı" test durumu için kullanılan özellik belirleme katsayısını, "Göç Sayısı" testte sonucunda karar verilen toplam göç sayısını , "Hesaplama Süresi(sn)" testin ne kadar sürdüğünü ifade eden alanlardır.

#### 5.3.2.2 Test2 ortamı

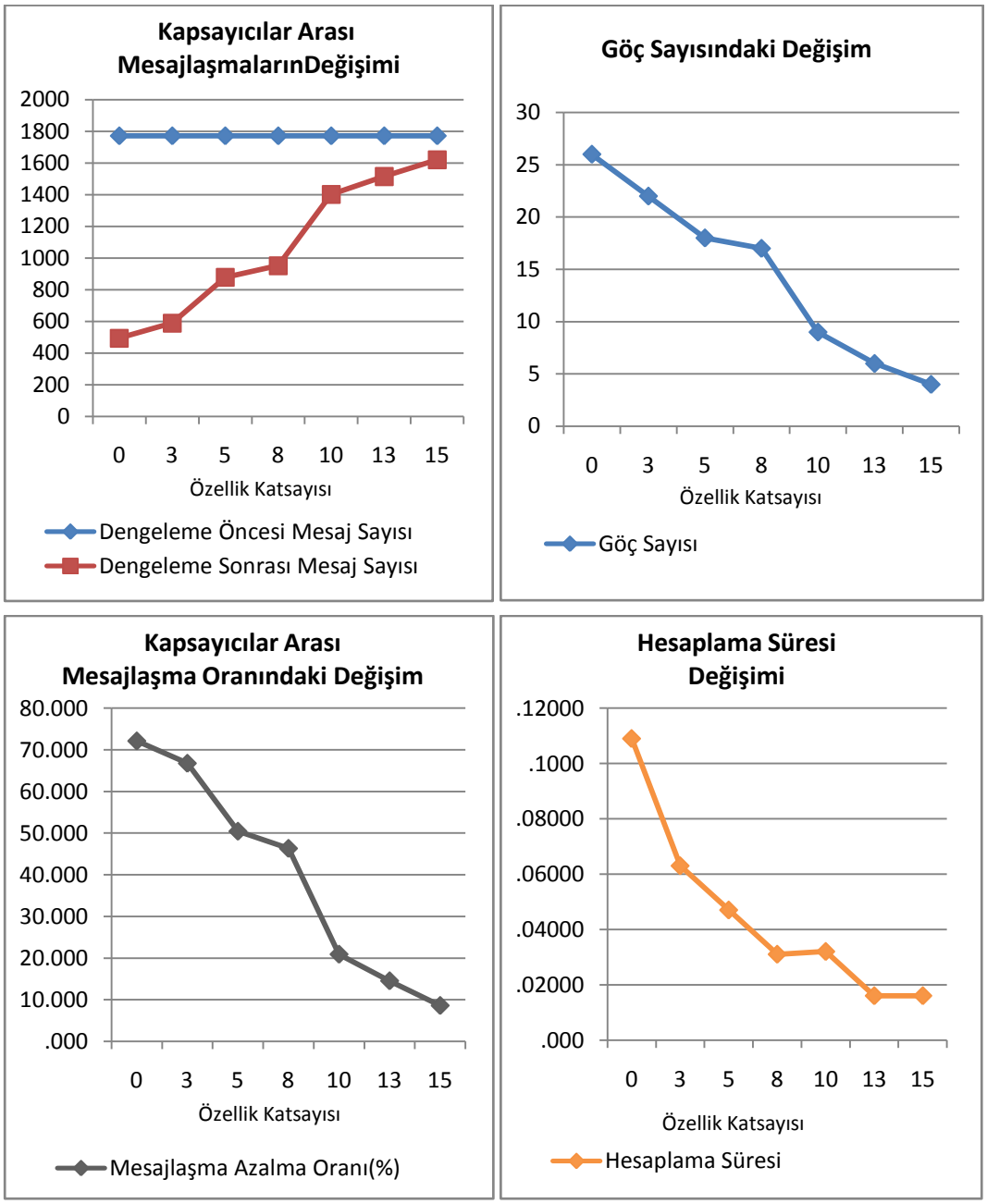
Test2 ortamında kapsayıcı sayısı 80, kapsayıcı başına düşen işçi sayısı 10, görevlere ait taşınma maliyetinin 19 ve kapsayıcı başına düşen görev sayısının 6 olduğu bir sistem ele alınmıştır. Test2 ortamına ait farklı mesajlaşma ve özellik katsayılarında yapılan testlerin sonuçları bölümün ilerleyen kısımlarında verilmiştir.

#### 5.3.2.3 Test3 ortamı

Test3 ortamında kapsayıcı sayısı 40, kapsayıcı başına düşen işçi sayısı 10, kapsayıcı başına düşen görev sayısının 6 olduğu bir sistem ele alınmıştır. Test3 testinde bir görev taşınma(göç ettirme) maliyetindeki değişimin algoritmamızın kararlarına etkisinin görülmesi amaçlanmıştır. Her bir taşınma maliyetinin değeri için yeni bir test ortamı üretilmektense önceki testlerden farklı olarak test ortamı bir kez üretilerek tüm taşınma maliyetleri için aynı mesajlaşma ve özellik durumlarında ağ trafik yükü dengeleme algoritmasının davranışı gözlemlenmiştir. Test ortamına ait farklı taşınma maliyetlerinde yapılan testlerin sonuçları aşağıda verilmiştir. Tüm test ortamlarında görevler arası mesajlaşma sayıları birim [0-80] kapalı aralığında rastgele olarak üretilmektedir. Daha önceki test ortamlarında taşınma maliyetinin sabit tutulmasından dolayı değinilmeyen bu durum bu ortam için önem arz etmektedir. Taşınma maliyeti 80 olarak belirlendiğinde test ortamlarımızda göç kararları algoritmamız tarafından verilememektedir. Bu sebeple test3 ortamında yapılan testlerimizde 75 taşınma maliyetine kadar algoritmamızın davranışını ele alacağız.

**Çizelge 5.1 :** Test1 Ortamında MK=0 iken yapılan testler

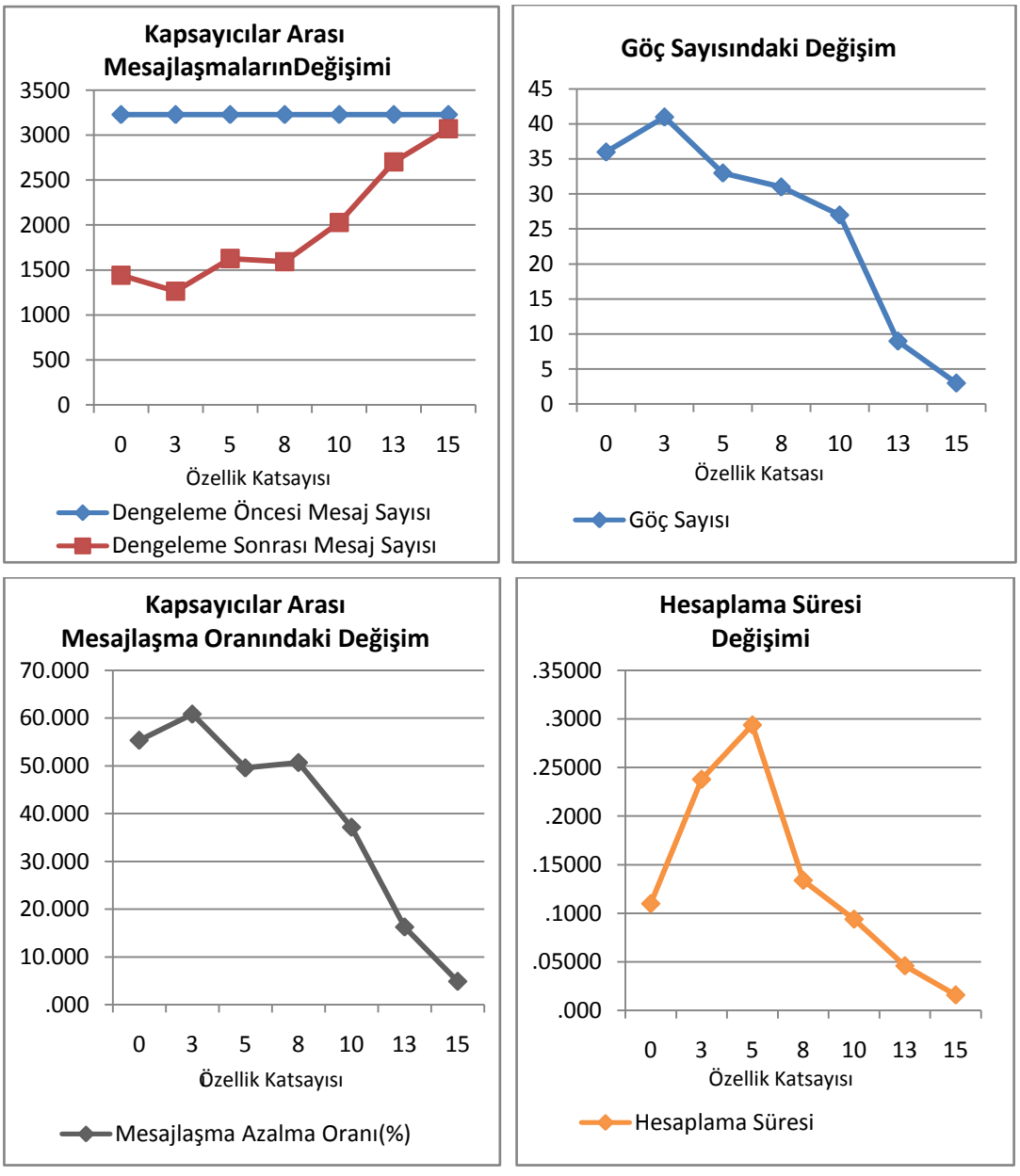
| Özellik Katsayısı | Göç Sayısı | Hesaplama Süresi | Dengeleme Öncesi Mesaj Sayısı | Dengeleme Sonrası Mesaj Sayısı | Mesajlaşma Azalma Oranı(%) |
|---|---|---|---|---|---|
| 0 | 26 | 0.11 | 1771 | 494 | 72.11 |
| 3 | 22 | 0.06 | 1771 | 589 | 66.74 |
| 5 | 18 | 0.05 | 1771 | 878 | 50.42 |
| 8 | 17 | 0.03 | 1771 | 951 | 46.30 |
| 10 | 9 | 0.03 | 1771 | 1401 | 20.89 |
| 13 | 6 | 0.02 | 1771 | 1514 | 14.51 |
| 15 | 4 | 0.02 | 1771 | 1619 | 8.58 |




**Şekil 5.17 :** Test1 ortamı mesajlaşma katsayısı:0 sonuç grafikleri

**Çizelge 5.2 :** Test1 Ortamında MK=5 iken yapılan testler

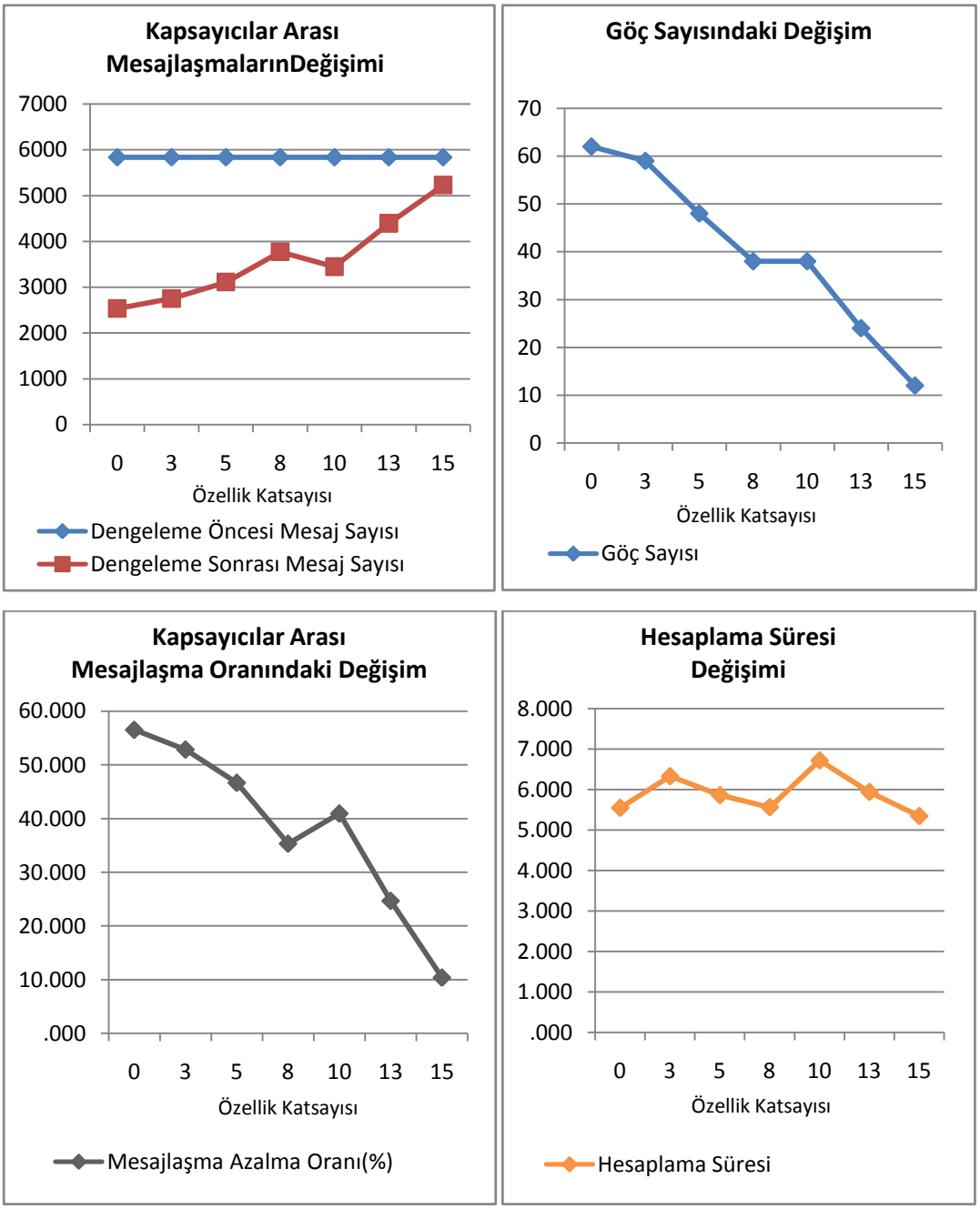
| Özellik Katsayısı | Göç Sayısı | Hesaplama Süresi | Dengeleme Öncesi Mesaj Sayısı | Dengeleme Sonrası Mesaj Sayısı | Mesajlaşma Azalma Oranı(%) |
|---|---|---|---|---|---|
| 0 | 36 | 0.1100 | 3229 | 1441 | 55.3732 |
| 3 | 41 | 0.2380 | 3229 | 1264 | 60.8548 |
| 5 | 33 | 0.2940 | 3229 | 1628 | 49.5819 |
| 8 | 31 | 0.1340 | 3229 | 1592 | 50.6968 |
| 10 | 27 | 0.0940 | 3229 | 2030 | 37.1322 |
| 13 | 9 | 0.0460 | 3229 | 2705 | 16.2279 |
| 15 | 3 | 0.0160 | 3229 | 3073 | 4.8312 |




**Şekil 5.18 :** Test1 ortamı mesajlaşma katsayısı:5 sonuç grafikleri

**Çizelge 5.3 :** Test1 Ortamında MK=10 iken yapılan testler

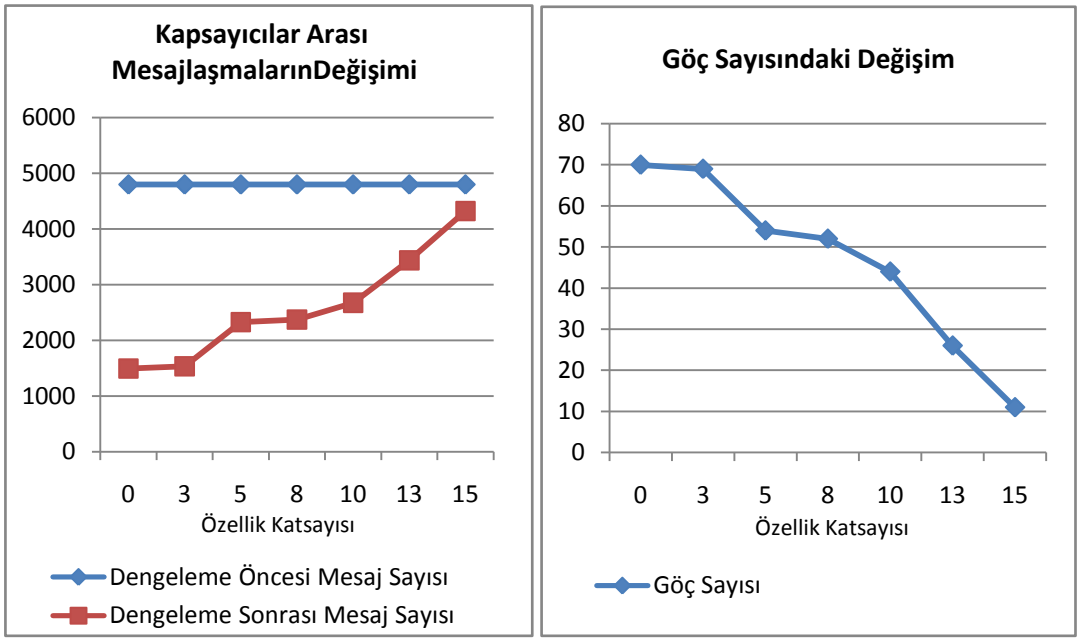
| Özellik Katsayısı | Göç Sayısı | Hesaplama Süresi | Dengeleme Öncesi Mesaj Sayısı | Dengeleme Sonrası Mesaj Sayısı | Mesajlaşma Azalma Oranı(%) |
|---|---|---|---|---|---|
| 0 | 62 | 5.5470 | 5834 | 2537 | 56.5135 |
| 3 | 59 | 6.3290 | 5834 | 2751 | 52.8454 |
| 5 | 48 | 5.8600 | 5834 | 3112 | 46.6575 |
| 8 | 38 | 5.5630 | 5834 | 3773 | 35.3274 |
| 10 | 38 | 6.7200 | 5834 | 3446 | 40.9325 |
| 13 | 24 | 5.9380 | 5834 | 4394 | 24.6829 |
| 15 | 12 | 5.3440 | 5834 | 5227 | 10.4045 |



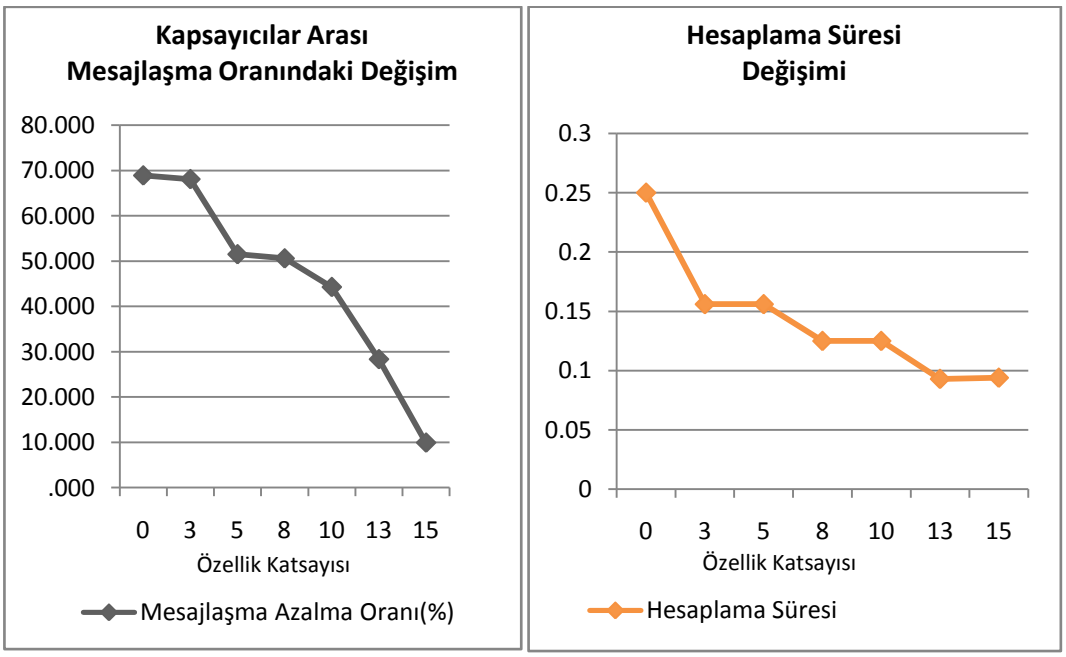

**Şekil 5.19 :** Test1 ortamı mesajlaşma katsayısı:10 sonuç grafikleri

**Çizelge 5.4 :** Test2 Ortamında MK=0 iken yapılan testler

| Özellik Katsayısı | Göç Sayısı | Hesaplama Süresi | Dengeleme Öncesi Mesaj Sayısı | Dengeleme Sonrası Mesaj Sayısı | Mesajlaşma Azalma Oranı(%) |
|---|---|---|---|---|---|
| 0 | 70 | 0.25 | 4800 | 1493 | 68.8958 |
| 3 | 69 | 0.156 | 4800 | 1532 | 68.0833 |
| 5 | 54 | 0.156 | 4800 | 2328 | 51.5000 |
| 8 | 52 | 0.125 | 4800 | 2372 | 50.5833 |
| 10 | 44 | 0.125 | 4800 | 2675 | 44.2708 |
| 13 | 26 | 0.093 | 4800 | 3440 | 28.3333 |
| 15 | 11 | 0.094 | 4800 | 4324 | 9.9167 |



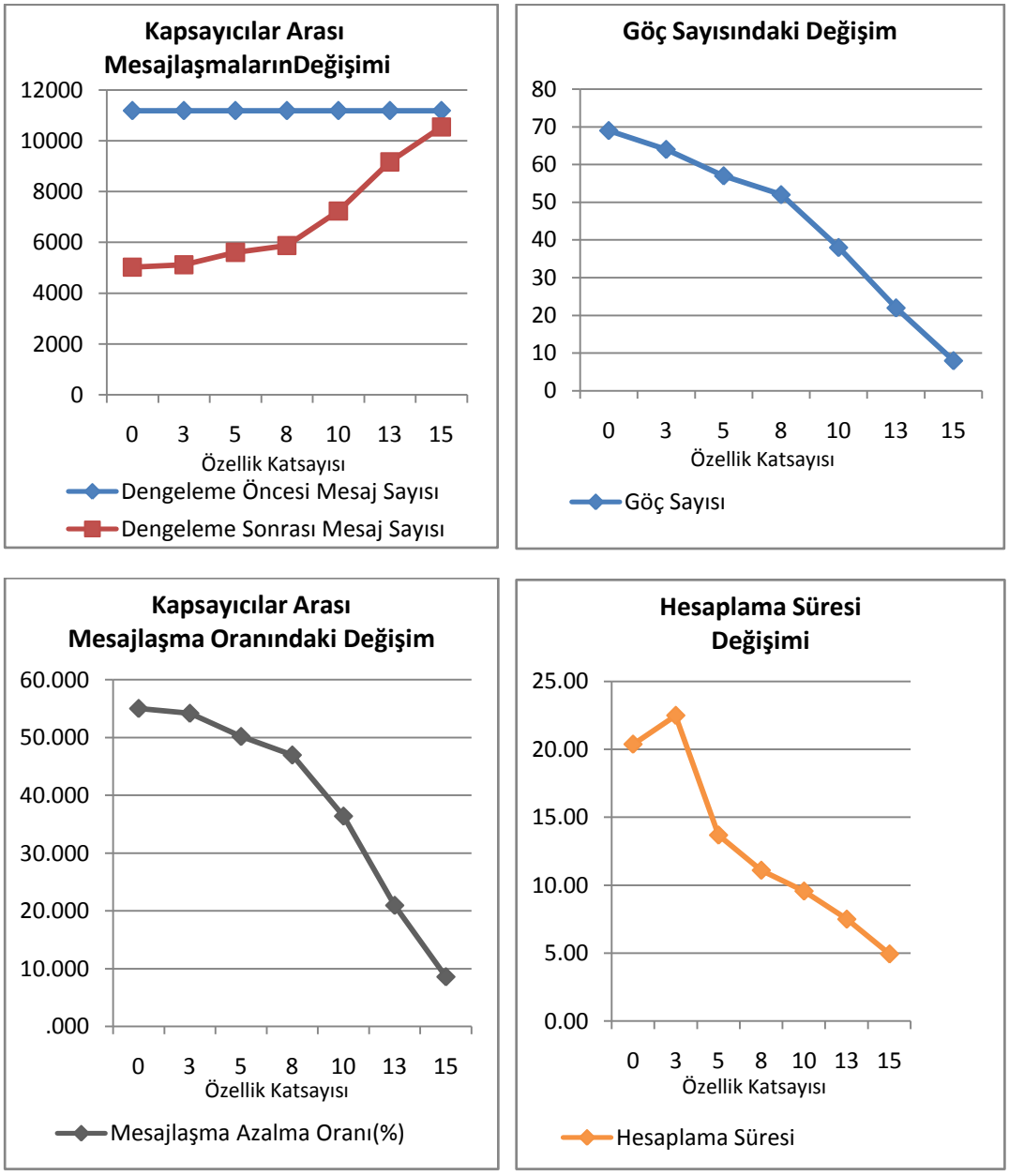

**Şekil 5.20 :** Test2 ortamı mesajlaşma katsayısı:0 sonuç grafikleri-1




**Şekil 5.21 :** Test2 ortamı MK=0 sonuç grafikleri-2

**Çizelge 5.5 :** Test2 Ortamında MK=5 iken yapılan testler

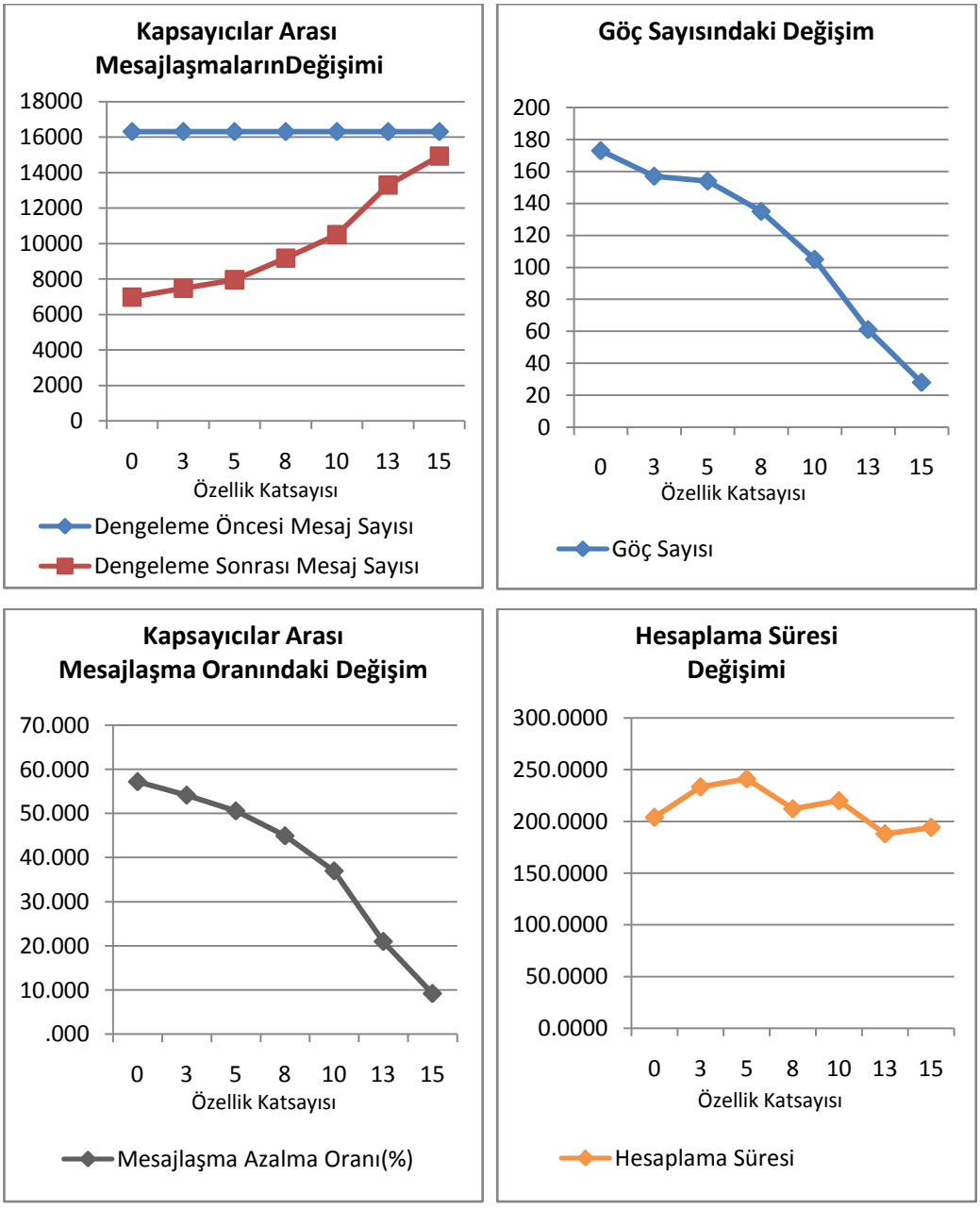
| Özellik Katsayısı | Göç Sayısı | Hesaplama Süresi(sn) | Dengeleme Öncesi Mesaj Sayısı | Dengeleme Sonrası Mesaj Sayısı | Mesajlaşma Azalma Oranı(%) |
|---|---|---|---|---|---|
| 0 | 131 | 20.37 | 11189 | 5031 | 55.00 |
| 3 | 127 | 22.48 | 11189 | 5126 | 54.16 |
| 5 | 113 | 13.69 | 11189 | 5614 | 50.15 |
| 8 | 107 | 11.10 | 11189 | 5880 | 46.95 |
| 10 | 83 | 9.57 | 11189 | 7235 | 36.37 |
| 13 | 46 | 7.51 | 11189 | 9171 | 20.93 |
| 15 | 19 | 4.95 | 11189 | 10543 | 8.57 |




**Şekil 5.22 :** Test2 ortamı MK=5 sonuç grafikleri

**Çizelge 5.6 :** Test2 Ortamında MK=10 iken yapılan testler

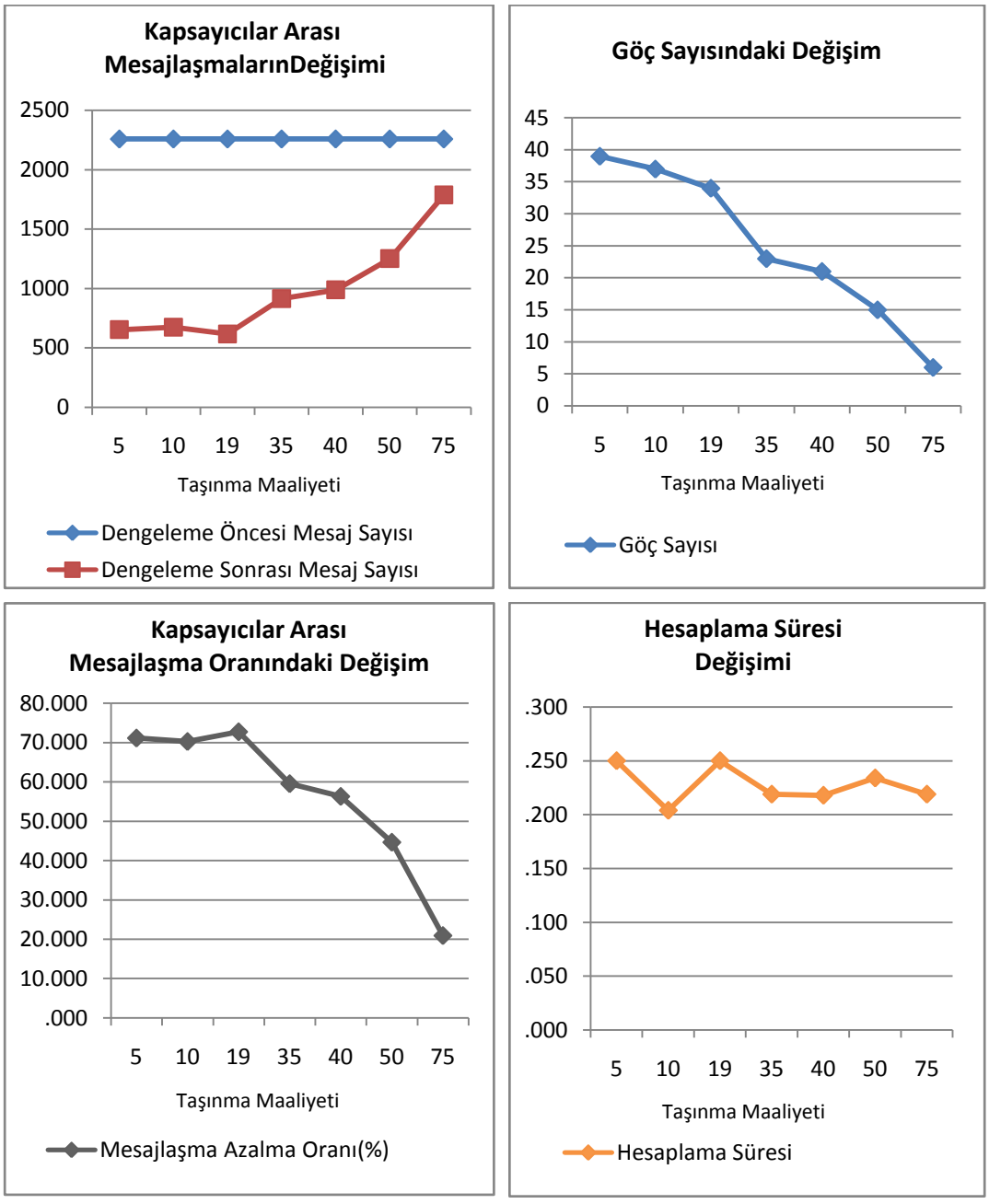
| Özellik Katsayısı | Göç Sayısı | Hesaplama Süresi | Dengeleme Öncesi Mesaj Sayısı | Dengeleme Sonrası Mesaj Sayısı | Mesajlaşma Azalma Oranı(%) |
|---|---|---|---|---|---|
| 0 | 173 | 203.7988 | 16308 | 6978 | 57.2218 |
| 3 | 157 | 233.3259 | 16308 | 7473 | 54.1787 |
| 5 | 154 | 240.9748 | 16308 | 7954 | 50.5913 |
| 8 | 135 | 211.9885 | 16308 | 9172 | 44.9498 |
| 10 | 105 | 219.8341 | 16308 | 10498 | 36.9986 |
| 13 | 61 | 187.8070 | 16308 | 13296 | 21.0072 |
| 15 | 28 | 194.0625 | 16308 | 14931 | 9.2172 |




**Şekil 5.23 :** Test2 ortamı MK=10 sonuç grafikleri

**Çizelge 5.7 :** Test3 Ortamında MK = 0 ve ÖK = 0 iken yapılan testler

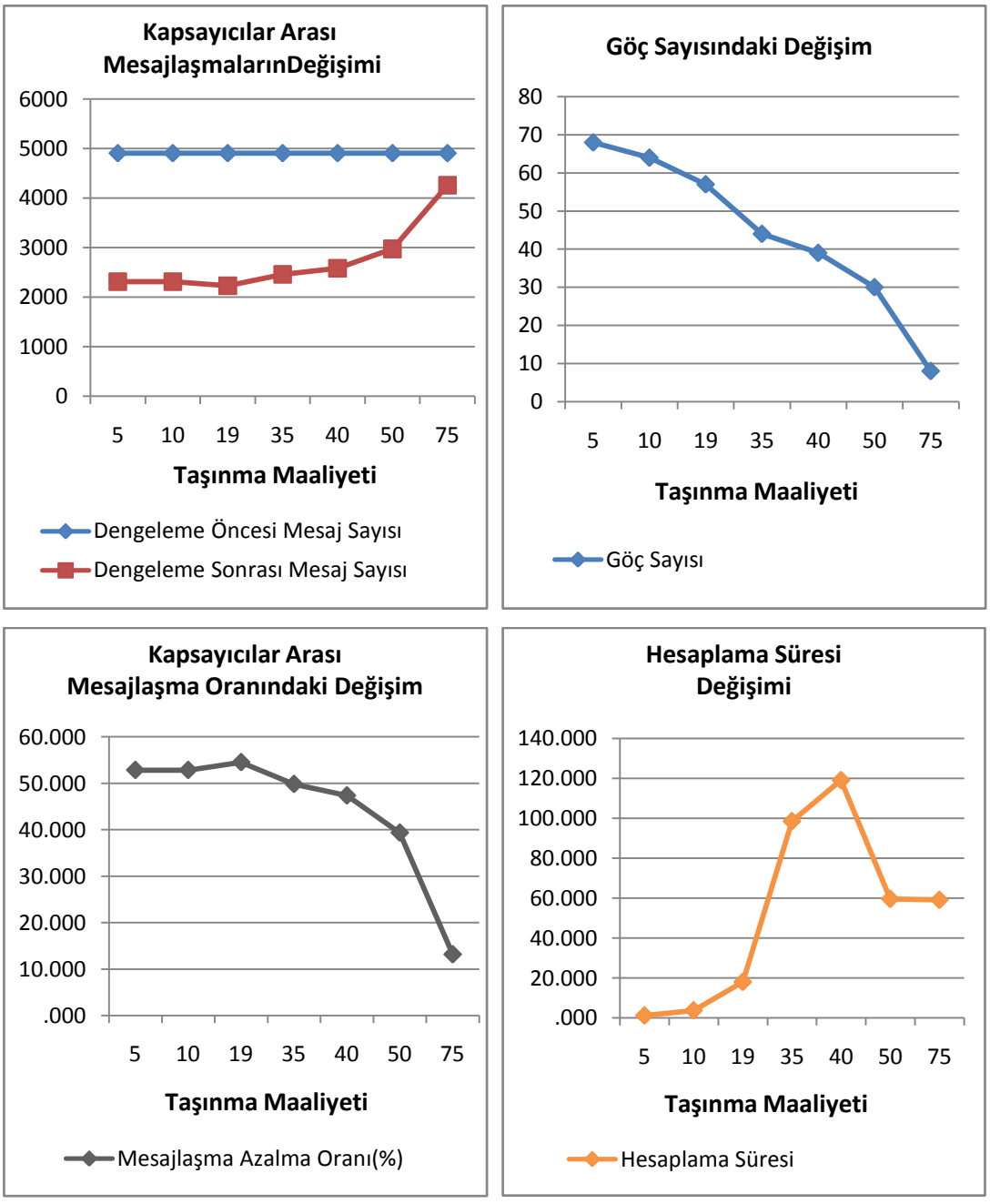
| Taşınma Maaliyeti | Göç Sayısı | Hesaplama Süresi | Dengeleme Öncesi Mesaj Sayısı | Dengeleme Sonrası Mesaj Sayısı | Mesajlaşma Azalma Oranı(%) |
|---|---|---|---|---|---|
| 5 | 39 | 0,2500 | 2258 | 652 | 71,1249 |
| 10 | 37 | 0,2040 | 2258 | 672 | 70,2391 |
| 19 | 34 | 0,2500 | 2258 | 616 | 72,7192 |
| 35 | 23 | 0,2190 | 2258 | 914 | 59,5217 |
| 40 | 21 | 0,2180 | 2258 | 987 | 56,2888 |
| 50 | 15 | 0,2340 | 2258 | 1250 | 44,6413 |
| 75 | 6 | 0,2190 | 2258 | 1786 | 20,9035 |




**Şekil 5.24 :** Test3 ortamı MK:0 ÖK:0 sonuç grafikleri

**Çizelge 5.8 :** Test3 Ortamında MK = 5 ve ÖK = 5 iken yapılan testler

| Taşınma Maaliyeti | Göç Sayısı | Hesaplama Süresi | Dengeleme Öncesi Mesaj Sayısı | Dengeleme Sonrası Mesaj Sayısı | Mesajlaşma Azalma Oranı(%) |
|---|---|---|---|---|---|
| 5 | 68 | 1,1870 | 4909 | 2315 | 52,8417 |
| 10 | 64 | 3,7340 | 4909 | 2315 | 52,8417 |
| 19 | 57 | 18,0150 | 4909 | 2232 | 54,5325 |
| 35 | 44 | 98,5470 | 4909 | 2463 | 49,8268 |
| 40 | 39 | 119,0780 | 4909 | 2584 | 47,3620 |
| 50 | 30 | 59,5940 | 4909 | 2977 | 39,3563 |
| 75 | 8 | 59,0930 | 4909 | 4263 | 13,1595 |




**Şekil 5.25 :** Test3 ortamı MK:5 ÖK:5 sonuç grafikleri

### 5.3.3 Sınama Ortamı ile Yapılan Test Sonuçlarının Değerlendirilmesi

Test1 ve Test2 ortamlarında gerçekleştirilen üç farklı deneme farklı rastgelelik katsayılarıyla üretildiklerinden, görevlerin özellik bağımlılıkları ve arama grafımızı oluşturan görevler arası mesajlaşmaların karmaşıklığı sonuçları etkileyen faktörler olarak göze çarpmaktadır. Test ortamlarında oluşan sonuçların rastgelelik faktöründen en az şekilde etkilenmesi için her bir test ortamı yaratılırken bütün farklı özellik ve mesajlaşma katsayısı çiftleri için 5'er kez algoritma çalıştırılarak çıkan sonuçların ortalaması alınmıştır. Test1 ve Test2 ortamlarına ait test sonuçları incelendiğinde görevlerin belirli özellikleri talep etmesi yoğunlaştığında görevler belirli kapsayıcılara zorunlu hale geldiklerinden çözüm hızı artmakta ancak göç sayısı azalmakta ve dolayısıyla kapsayıcılar arası mesajlaşma sayısındaki azalma oranı düşmektedir. Mesajlaşma sayıları arttırıldığında ve görevlerin özellik talep etme oranı azaltıldığında çözüm zamanı uzamakta ve göç sayısı artmaktadır.

Test2 ortamında yer alan en yoğun mesajlaşmanın olduğu(MK=10) test durumunda görevler arasındaki mesajlaşmaların yoğunluğu ve görev sayısı fazla olduğundan bu örnekte tüm sistemdeki görevleri alt gruplara ayırarak çözümleme yapma algoritmamız yoğun olarak çalışmaktadır. Bu durumun çözümünde oluşturulan alt görev gruplarının büyüklüğü ve bu gruplara denk gelmiş olan özellikle sabitlenmemiş görev sayıları çözüm süresinde belirleyici olmaktadır. Mesajlaşma ağı yoğun olduğunda alt gruplarda özellikle bir kapsayıcıya bağlanmış görev sayıları yakın çıktığından bu örnekte tüm özellik katsayılarında üretilen test durumları için çözüm zamanı yakın olarak çıkmaktadır.

Test3 ortamı ele alındığında görev göç maliyetlerindeki artışın gerçekleşen göç sayısını azalttığı görülmektedir. Aynı şekilde görev göç maliyeti düşürüldüğünde ise gerçekleşen göç sayıları artmakta ve dolayısıyla kapsayıcılar arası mesajlaşma sayıları azalmaktadır. Taşınma maliyetindeki değişimlerin, hesaplama süresi üzerimde anlamlı bir etkisi gözlemlenememiştir.

Sonuç olarak yapılan testler sonucunda görülmektedir ki bir özellikle kapsayıcılara bağlanmamış her bir görev çözüm aranacak arama uzayını üstel olarak büyütmektedir ve çözüm zamanını uzatmaktadır. Bu sebeple testlerimizde ne kadar fazla sayıda serbest görev bulunuyorsa çözüm zamanımız o kadar uzamaktadır. Ayrıca görevlerin taşınma maliyetlerinin azaltılması beklenildiği gibi verilen göç kararı sayısını arttırmaktadır.

# 6. SONUÇ ve ÖNERİLER

Tezimizde etmen tabanlı bir grid sisteminde güçlü taşınabilirlikli kod göçü kullanarak ağ trafik yükü dengelemesi yapılmaktadır. Etmen sistemlerinin esnek, dinamik yapıları ile grid sistemlerinin güçlü, kararlı yapılarının birleştirilmesiyle oluşturulmuş etmen tabanlı bir grid sistemine, esnek bir ağ trafiği yükü dengeleme algoritması ve çalışan görevlerin bağlam bilgilerinin korunarak taşınmasını sağlayacak görev göçü gerçekleştirebilme yeteneği eklenmiştir.

Tezimizde kullandığımız etmen tabanlı grid sisteminde [1] işlemci kaynağı paylaşımı işçilere görev ataması yapılırken paylaştırılmaktaydı. İşçi etmenlere görev atamaları yapılırken görev atamaları sırasındaki işlemci yükü göz önünde bulundurulmakta ve bu şekilde çalışmaya başlayan görevler için çalışma zamanında bir yük dengeleme mekanizması bulunmamaktaydı. Tezimiz kapsamında bu sisteme çalışma zamanında yük dengeleme yapabilecek bir mekanizma eklemeyi amaçladık. Sistemimizde çalıştırdığımız görevlerin işlemci yüklerinin çok büyük olmaması, görev işletimlerinin işlemci kullanım yüzdesine etkisinin çok küçük olması ve işlemciye ait çeşitli sistem kullanımı bilgilerinin sistem bağımlı yollarla edinilebilmesi gibi sebeplerle, gerçeklediğimiz sistemde yönetiminin ve ölçülmesinin nispeten kolay olacağını düşündüğümüz ve dağıtılmış sistemlerin yumuşak karınlarından biri olan ağ trafiğini dengelenecek yük olarak belirledik. Ağ yükünün yönetimi ve ölçülmesi şu açılardan kolay olabilmektedir. Ağ yükünü JADE mimarisinde yer alan farklı kapsayıcılar arasında gerçekleşen mesajlaşma sayılarına endeksleyebilmekteyiz. Bu sayede bu mesajlaşmaların sayısı bize ağ yükü ile ilgili fikir vermektedir. Bununla beraber etmenler arası mesajlaşma sayılarını istediğimiz gibi değiştirebilmemiz ise ağ yükünü kolayca yönetebilmemizi sağlamaktadır.

Yük dengeleme algoritmamız bilgisayar bilimlerinden yapay zekanın ilgi alanına giren kısıt karşılama problemlerini temel almaktadır. Tezimizin asıl ilgi odağı olarak gördüğümüz ve geliştirme sırasında üzerinde en çok durulan kısım bir grid sistemine ait yük dengeleme algoritmasının bir kısıt karşılama problemine dönüştürülerek çözülebileceği önermesidir. Tezimizin diğer kısımlarında yer alan etmen tabanlı grid

sistemi ve güçlü taşınabilirlikli görev göçlerinin gerçeklenmesi bu önermenin denenmesi için seçilmiş uygun bir uygulama alanı olarak öne çıkmaktadır.

Yük dengeleme algoritması kapsamında, sistemimiz açısından yük olarak düşündüğümüz öğeleri kısıt olarak ekleyerek oluşturduğumuz kısıt problemini açık kaynak kodlu ve Java tabanlı bir kısıt çözüm kütüphanesi kullanarak çözmekteyiz. Bu şekilde tasarladığımız yük dengeleme algoritması oldukça esnek bir yapı arz etmektedir. Sistemimiz açısından yük olarak düşüneceğimiz herhangi bir öğeyi rahatlıkla kısıt olarak problemimize ekleyerek yük dengeleme algoritmasını değiştirebilmekteyiz. Örneğin bir kapsayıcıdaki işlemci kullanımının artmasının sistemdeki toplam maliyeti arttırması gibi bir durumun uygun bir şekilde kısıt haline getirilip problemimize eklenmesiyle yük dengeleme algoritmamız kolaylıkla işlemci yükünü hesaba katarak karar verebilen bir hale getirilebilir bir yapıdadır.

Yük dengeleme için görevleri güçlü taşınabilirlikli bir şekilde bağlam bilgileriyle göç ettirmemiz görev tarafından yapılan işletimin ve elde edilen verilerin kaybolmasını engellemektedir. Sistemimizde çalıştırdığımız test görevleri açısından bağlam bilgisinin korunmasının önemi yoktur. Ancak canlı, esnek ve etkileşimlerin yüksek olduğu bir sistemde çevresini gözlemleyen ve çevresiyle sürekli iletişim halinde olan bir görev için uzun bir süre çalıştıktan sonra kendi içerisinde tuttuğu değişkenlerin değerleri ve program akışının geldiği yer önemlidir. Bu değerlerin ve program akışının olması gereken yere gelmesi için etmenin aynı mesajları göndermesi, alması ve aynı etkileşimlerde bulunması gereklidir. Böyle bir durumda görevlerin değişken değerlerini ve program akışını kaybetmemesinin en kolay yolu bağlam bilgisini korumaktır. Tezimizde bu tip görevler için çözüm yolu önerilmiştir ve güçlü taşınabilirlikli görev göçü gerçeklenmiştir.

Tezimizde kullanılan tüm kütüphaneler ve araçlar Java tabanlıdır. Sistemin Java tabanlı bir yapıda olması platform bağımsız bir yapıda olmalarını sağlamaktadır. Her ne kadar reel olarak ihtiyaç olmayacak bir durum olsa da farklı platformlar arası görev göçü sağlamak mümkün olabilmektedir. Bu da tezimizde önerdiğimiz sistemin sunduğu bir başka esneklik olarak görülebilir.

Test sonuçlarına göre yük dengeleme algoritmamızın verdiği kararlar ve gerçekleştirdiği görev göçleri sayesinde kapsayıcılar arasındaki mesajlaşma sayılarını azaltarak ağ yükünü dengelediği görülmektedir. Sınama aracımız ile oluşturulan sanal etmen tabanlı grid sistemleri için farklı durumlar oluşturulup test

edilebilmektedir. Oluşturulan sanal sistemler için bir görevin göç ettirilme maliyeti ve görevler arası mesajlaşma sayıları kullanıcı ara yüzleri yardımıyla kolaylıkla değiştirilip, oluşacak durumlar karşısında yük dengeleme algoritmamızın verdiği kararlar ve bunun ağ yüküne etkileri kolaylıkla gözlemlenebilmektedir.

## 6.1 Öneriler ve İleriki Çalışmalar

Gerçeklediğimiz sistem esnek bir yük dengeleme algoritması sunmaktadır. Ancak yük dengeleme algoritması için bahsettiğimiz şekilde yeni kısıtlar eklenerek algoritmanın geliştirilmesi işlemi uygulama kodu içerisinde ilgili kısım bulunup burada güncelleme yapılarak gerçekleştirilmektedir. Esnek ve kolay anlaşılabilir bir yapısı olmasına rağmen sistemin dışarıdan kullanılması kısıtlı ve uygulamamızın koduna müdahale ile mümkündür. Göç yönetici etmenin sunacağı bir ara yüz ile sistemin yük dengeleme algoritması için belirlenecek kısıtların tezimizde yapmış olduğumuz gerçeklemelerden soyut olarak yük dengeleme algoritmasına eklenebilir olmasının sağlanması, sistemin yeniden kullanılabilirliği ve esnekliği açısından olumlu olacaktır.

Çalışmamız kapsamında etmen tabanlı grid sistemini ve yük dengeleme algoritmamız çok yüksek sayıda kapsayıcı ve işçi etmeni olan yüksek bir dağıtılmışlığa sahip bir sistem için denenmemiştir. Çok yüksek sayıda kapsayıcı, etmen ve görev çalıştırma potansiyeli olabilecek bir sistem için tek bir göç yönetici etmen yeterli gelmeyeceği öngörülebilir. Bu sebeple bu tarz sistemler için sistemin ihtiyacına göre eklenecek ve birbirleriyle koordineli şekilde yük dengeleme mekanizmalarını gerçekleştirebilecek çok sayıda göç yönetici etmenin sisteme eklenmesi gerekebilir. Sistemin bu yönde geliştirilmesi büyük sistemlerde kullanılabilmesine olanak sağlayacaktır.

## KAYNAKLAR


[1] **U. Gümüş,** 2009. Etmen Tabanlı Bir Grid Sistemi, Yüksek Lisans Tezi, İTÜ Bilgisayar Mühendisliği Bölümü.

[2] **I. Foster, C. Kesselman, J. Nick, S. Tuecke,** 2002. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum*, June 22, 2002.

[3] **I. Foster, Nicholas R. J., C. Kesselman,** 2004. Brain Meets Brawn: Why Grid and Agents Need Each Other. *Proceedings of 3rd Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, New York, USA 2004.

[4] **Truyen, E., Robben, B.,Vanhaute, B., et al.** 2000 : Portable support for transparent Thread migration in Java. In Procedings of ASA/MA2000.

[5] **Jacop,** http://jacopguide.osolpro.com/guideJaCoP.html

[6] **Jacob, B. Brown, M. Fukui, K. Trivedi N.**, 2005. Introduction to Grid Computing. IBM.

[7] **The EDUCAUSE Learning Initiative,** 2006. 7 Things You Should Know About Grid Computing, ELI 7 Things You Should Know, EDUCAUSE Learning Initiative.

[8] **I. Foster, C. Kesselman, S. Tuecke,** 2001. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[9] **Luck M., McBurne P., Preist** C., 2003. Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing). AgentLink. ISBN 0854 327886.

[10] **Jennings, N. R., Sycara, K., and Wooldridge, M.,** 1998. A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems 1, 1 (Jan. 1998), 7-38.

[11] **Yoon Y., Lee G., Choi K., and Shin D.,** 2008. Design of Agent Service System to Manage Services among Heterogeneous Multi-agent Systems. Proceedings of the 2008 IEEE international Workshop on Semantic Computing and Applications - Volume 00 (July 10 - 11, 2008). IWSCA. IEEE Computer Society, Washington, DC, 123-125.

[12] **FininT., Fritzson R., McKay D., and McEntire R.,** 1994. KQML as an agent communication language. Proceedings of the Third international Conference on information and Knowledge Management (Gaithersburg, Maryland, United States, November 29 - December 02,1994). N. R. Adam, B. K. Bhargava, and Y. Yesha, Eds. CIKM '94. ACM, New York, NY, 456-463.

[13] **Foundation for Intelligent Physical Agents,** 2002. FIPA ACL Message

Structure Specification.

[14] **Fabio B., Giovanni C., Tiziana T. Giovanni R.,** 2007. JADE Programmer's Guide.

[15] **U. Gümüş, N.Erdoğan,** 2010 AGRID - Agent Based Grid System. *1st International Workshop on Infrastructures and Tools for Multiagent Systems ( ITMAS Workshop at AAMAS 2010),* Toronto 2010.

[16] **A. Genç, ,** 2010 "Strong Migration of Java Threads", Yüksek Lisans Tezi, İTÜ Bilgisayar Mühendisliği Bölümü.

[17] **Fünfrocken,S.** 1998: "Transparent Migration of Java-based Mobile Agents" In MA'98 Mobile Agents, volume 1477 of LNCS,pages 26-37.Springer-Verlag.

[18] **S. Bouchenak, D. Hagimont, S. Krakowiak, N. D. Palma, and F. Boyer,** 2002. Experiences Implementing Efficient Java Thread Serialization, Mobility and Persistence. *Technical Report RR-4662,* INRIA, December 2002.

[19] **R. S. Gray, G. Cybenko, D. Kotz, R. A. Peterson, and D. Rus,** 2002. D'Agents: Applications and Performance of a Mobile-Agent System. *Software Practice and Experience*, 32(6), May 2002.

[20] **A. Acharya, M. Ranganathan, and J. Saltz.,** 1996. Sumatra: A Language for Resource-Aware Mobile Programs. In Mobile Object Systems: Towards the Programmable Internet, number 1222 in Lecture Notes in Computer Science. Springer-Verlag.

[21] **T. Suezawa,** 2000. Persistent execution state of a Java virtual machine. *In Proceedings of the ACM 2000 conference on Java Grande*, 2000.

[22] **B. Overeinder, N. Wijngaards, M. van Steen, and F. Brazier,** 2002. Multi-Agent Support for Internet-Scale Grid Management. *In AISB'02 Symposium on AI and Grid Computing, April 2002*.

[23] **S. Funfrocken,** 1998. Transparent Migration of Java-based Mobile Agents: Capturing and Reestablishing the State of Java Programs. In Proceedings of the Second International Workshop on Mobile Agents, Stuttgart, Germany, September 1998.

[24] **T. Sekiguchi, H. Masuhara, and A. Yonezawa**, 1999. A Simple Extension of Java Language for Controllable Transparent Migration and its Portable Implementation. In Coordination Models and Languages, 1999.

[25] **Sakamoto T., Sekiguchi, T., and Yonezawa,** A. 2000 : Bytecode transformation for portable Java Thread Migration in Java. In Procedings of ASA/MA2000.

[26] **Coninx T., Truyen E.,Vanhaute B., et al.** 2000 : On the use of threads in mobile object systems . InWorkshops of ECOOP 2000.

[27] **S. Russell and P. Norvig.** 2002 : Artificial Intelligence: A Modern Approach (2nd Edition) 137-140.

[28] **DistriNet**, http://distrinet.cs.kuleuven.be/

**ÖZGEÇMİŞ**

**Ad Soyad: Zafer Altuğ Sayar**

**Doğum Yeri ve Tarihi: Trabzon- 28.04.1986**

**Adres: Tatlısu Mah. Tuncay Sok. No:3 Daire:21 Ümraniye/İstanbul**

**E-Posta: zaferaltug@hotmail.com**

**Lisans: İstanbul Üniversitesi Bilgisayar Mühendisliği (2008)**

**Mesleki Deneyim,Yayınlar ve Ödüller:**

**2008 - .. Tübitak Bilgem Bilişim Teknolojileri Enstitüsünde Araştırmacı**

**2012 - Sayar, Z. A., & Erdoğan, N. Network Load Balancing with Strong Migration in an Agent Based Grid System Using CSP Approach. International Journal, 3.**