T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

JSTAR – KAYNAK BAĞIMLI UYGULAMALARIN CLUSTER DÜZENİNDE ÇALIŞTIRILMASI İÇİN JAVA FRAMEWORK

YASİN KAYA

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMAN
YRD. DOÇ. DR. A. GÖKHAN YAVUZ

İSTANBUL, 2011

T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

JSTAR – KAYNAK BAĞIMLI UYGULAMALARIN CLUSTER DÜZENİNDE ÇALIŞTIRILMASI İÇİN JAVA FRAMEWORK

YASİN KAYA

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMAN
YRD. DOÇ. DR. A. GÖKHAN YAVUZ

İSTANBUL, 2011

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

JSTAR – KAYNAK BAĞIMLI UYGULAMALARIN CLUSTER DÜZENİNDE ÇALIŞTIRILMASI İÇİN JAVA FRAMEWORK

Yasin KAYA tarafından hazırlanan tez çalışması 20.12.2011 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Yrd. Doç. Dr. A. Gökhan Yavuz

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Yrd. Doç. Dr. A. Gökhan Yavuz

Yıldız Teknik Üniversitesi _____

Prof. Dr. A. Coşkun Sönmez

Yıldız Teknik Üniversitesi _____

Prof. Dr. Bülent Örencik

İstanbul Teknik Üniversitesi _____

ÖNSÖZ

Gerçekleştirilen çalışmanın konusunu, kaynak bağımlı uygulamaların küme düzeninde çalıştırılabilmesi için kaynak sisteme erişimi kontrol eden ve bunu adaptörler aracılığı ile daha da kolaylaştıran bir java çatı çözümünün geliştirilmesi oluşturmaktadır. Ayrıca geliştirilen çatı çözümünü, gerçek kaynak sistemler üzerinde kullanarak elde edilen kazanımın ve test sonuçlarının analiz edilmesi amaçlanmıştır.

Bu çalışmayı gerçekleştirmem konusundaki sabır ve desteklerinden dolayı aile bireylerime, cesaretlendirici ve yönlendirici tavırlarından dolayı başta tez danışmanım değerli hocam Yrd. Doç. Dr. A. Gökhan YAVUZ olmak üzere tüm Yıldız Teknik Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı öğretim üyelerinine sonsuz teşekkür ediyorum.

Haziran, 2011

Yasin KAYA

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

A/A	Kümede bulunan her iki düğümün de aynı anda aktif olması
A/P	Kümede bulunan düğümlerden birinin aktif, diğerinin pasif olması

KISALTMA LİSTESİ

CPU	Central Processing Unit
DAO	Data Access Object
DB	Database
DNS	Domain Name Service
FTP	File Transfer Protocol
HA	High-Availability
HTTP	Hypertext Transfer Protocol
ID	Identity
IP	Internet Protocol
J2ME	Java Platform, Micro Edition
JAR	Java ARchive
JDK	Java Development Kit
JEE	Java Platform, Enterprise Edition
JRE	Java Runtime Environment
JVM	Java Virtual Machine
OGSA	Open Grid Services Architecture
P2P	Peer-To-Peer
RAM	Random Access Memory
SPOF	Single Points Of Failure
SQL	Structured Query Language
SPSS	Statistical Package for the Social Sciences
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
VO	Value Object

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖZET

JSTAR – KAYNAK BAĞIMLI UYGULAMALARIN CLUSTER DÜZENİNDE ÇALIŞTIRILMASI İÇİN JAVA FRAMEWORK

Yasin KAYA

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Yrd. Doç. Dr. A. Gökhan YAVUZ

Donanım ve yazılım dünyasında yaşanan hızlı gelişmelere paralel olarak uygulama geliştiricilerin ve uygulama kullanıcıların beklentileri de artmıştır. Günümüzde uygulamaların yalnızca başarılı bir şekilde çalışabiliyor olması ihtiyaçları karşılamakta yetersiz kalmaktadır. Geliştirilecek olan bir uygulamanın performansı, hızı, ölçeklenebilirliği, kesintisiz hizmet verebilirliği gibi özellikler artık olmazsa olmazlar arasına girmiştir.

Bu çalışma kapsamında geliştirilen JStar Java çatısı bütün bu ihtiyaçların karşılanabilmesinde gerekli olan alt yapıyı sunmak için tasarlanmış bir java çatı çözümüdür. Uygulama geliştiricileri uygulamanın dışında kalan bu konulardan soyutlayarak, geliştirilecek olan uygulamanın iş mantığına odaklanmalarını sağlamak asıl hedef olarak belirlenmiştir. Platform olarak Java'nın seçilmiş olması tesadüfi bir karar olmayıp Java'nın ve açık kaynak dünyasının sunmuş olduğu imkanlardan yararlanmak, platform bağımsız uygulamalar geliştirmek ve Java'nın yaygın kullanım ağının getirilerinden azami derecede istifade etmek amaçlanmıştır.

Anahtar Kelimeler: Küme uygulamalar, java, kaynak sistem erişim kontrolü, çatı, yük dengeleme

ABSTRACT

JSTAR – A JAVA FRAMEWORK TO RUN RESOURCE DEPENDENT APPLICATIONS AT CLUSTER SYSTEM

Yasin KAYA

Department of Computer Engineering

MSc. Thesis

Advisor: Assist. Prof. Dr. A. Gökhan YAVUZ

In parallel with the development in hardware and software areas, expectations of application developers and application users have been increased rapidly. Nowadays, the application which only works fine is not sufficient to fulfill the requirements. Features like performans, speed, scalability, uninterrupted working are must for any application.

JStar Java Framework, which is developed in the scope of this thesis, is a java framework that offers needed infrastructure to fulfill these requirements. The main goal is to make application developers to concentrate on the application business logic by isolating them from these requirements which are out of the concept of the application. In this study, choosing java is not an incidental decision. Benefiting from open-source world, developing platform independent applications and taking advantages of commenly used java network are the main criteria of our decision.

Key words: Cluster applications, java, control of accessing resource system, framework, load balancing

BÖLÜM 1

GİRİŞ

1.1 Literatür Özeti

Günümüz yazılım dünyasında bir uygulamanın başarılı bir şekilde çalışıyor olması olmazsa olmaz bir şart olmakla birlikte tek başına yeterli değildir. Gerek geliştirilen uygulamaların müşterileri konumunda olan firmalar açısından, gerekse uygulamanın tüketicileri olan son kullanıcılar açısından bakıldığında bir uygulamaya zamanın her anında kesintisiz ve sürekli bir şekilde erişilebiliyor olması elzemdir. Aynı şekilde uygulamaların yüksek yük altındaki performansları, yapılan isteklere üretilen cevap süreleri uygulamaların başarılarının ölçümünde kullanılan temel kriterler olmuştur.

Bu durum yazılım dünyasına aşağıdaki kavramların girmesine neden olmuştur:

- Kümeleme
- Yük dengeleme
- Yüksek bulunurluk ve yedeğe geçme
- Ölçeklenebilirlik
- Dağıtımlı hesaplama
- Artıklık ve hataya dayanıklılık

Bu kavramların yazım dünyasına kazandırdıkları aslında özetle şu şekildedir; dışarıdan bakıldığında tek bir sistem vardır fakat bu sistem kendi içerisinde çok daha komplike bir yapıya sahiptir ve temel amaç sistemin kesintisiz hizmet verebilmesi, dengeli yük dağılımı yapabilmesi ve yüksek performanstır [1].

1.2 Tezin Amacı

Günümüz yazılım dünyasının temel ihtiyaçlarından biri olan bir uygulamanın aynı anda birden fazla makinede çalıştırılabilmesi, bu çalışmanın temel amacını oluşturmaktadır. Özellikle kaynak sistem bağımlı uygulamaların aynı anda birden fazla örnek olarak çalıştırılması bir kaydın birden fazla işlenmesi (mükerrer kayıt işleme) gibi çok ciddi problemler oluşturmaktadır. Bu çalışma kapsamında geliştirilen JStar çatısı bu sıkıntının çözülmesine yönelik bir altyapı çözümüdür.

Ayrıca 4 farklı kaynak sistem erişimi (dosya sistemi, veritabanı, ftp ve http) için geliştirilen adaptörler ile uygulama geliştiricilerin çatıları kullanmaktaki sıkıntılarından kaçınılmak amaçlanmıştır. Bu adaptörler sayesinde uygulama geliştirici mümkün olduğunca çatının yapısından soyutlanmıştır. Böylelikle geliştiriciler geliştirmekte olduğu uygulamaya daha fazla konsantre olma imkanı bulacaklardır.

1.3 Orjinal Katkı

JStar çatısı uygulama geliştiricilere yönelik bir çözüm olup geliştirilecek olan uygulamanın aynı anda birden fazla örnek olarak çalıştırılmasını sağlamaktadır. Son zamanlarda giderek daha da çok ilgi çeken kümele, dağılımlı hesaplama, yük dengeleme, yedeğe geçme gibi kavramlar JStar çatısı içerisinde de yer bulmuşlardır. Bu kavramların ortaya koymuş olduğu mantalite çalışma boyunca temel amaç olarak değerlendirilmiş ve çözümler bu şekilde üretilmiştir.

BÖLÜM 2

KÜMELEME

2.1 Dünden Bugüne Yazılım Dünyası

Yazılım, uygulama geliştiricilerin belirli araçlar kullanarak geliştirdiği ve belirli bir donanım üzerinde çalışan makine komutları dizisine verilen genel bir isimdir. Yazılım her şeyden önce üzerinde çalışacağı bir platforma ihtiyaç duymaktadır ve yetenekleri de yine bu platformun sunmuş oldukları ile sınırlıdır. Yazılım dünyası ilk çıktığı yıllardan günümüze gelinceye kadar birçok evreden geçmiştir. Çizelge 2.1'de bu durum kronolojik olarak gösterilmiştir.

Çizelge 2. 1 Programlama dillerinin tarihi gelişimi [2]

KUŞAK	PERİYOT	AÇIKLAMA
Birinci kuşak	1943 - 1958	Makine seviyesi programlama dilleri. 0 ve 1'lerden oluşur. Yazmak, okumak ve debug etmek çok zordur. Doğrudan CPU üzerinde çalışır ve çok hızlıdır
İkinci kuşak	1959 - 1964	Çevirme dilleri. Çevirici aracılığı ile yazılan kod makine koduna dönüştürülür
Üçüncü kuşak	1965 – 1970	Yüksek seviyeli diller; fortran, lisp, cobol, algol vb…
Dördüncü kuşak	1971 – günümüz	Sorgulama dilleri, rapor üreticiler, spesifikasyon dilleri; Maple, Postscript, SPSS, SQL

Beşinci kuşak	gelecek	Yapay zeka dilleri; prolog

Yazılımın üzerinde çalıştığı platform olan donanımda gerçekletirilen gelişmeler doğrudan yazılım dünyasını da etkilemiş, gerek uygulama geliştiriciler ve gerekse uygulama kullanıcılar bu gelişimin hızlanmasında katalizör görevi görmüşlerdir. Uygulama geliştiriciler donanımın sunmuş olduğu imkanlardan daha fazla yararlanmak ve uygulamalarını gerek rakiplerine göre ve gerekse bir önceki uygulamalarına göre daha tercih edilebilir hale getirmek için büyük bir yarışın içerisine girmişlerdir. Aynı şekilde uygulama tüketicileri olan son kullanıcılar da daha hızlı, daha güvenli, daha kullanım kolaylığı yüksek olan uygulamara rağbet göstermişlerdir.

Bu kapsamda yeni nesil programlama dilleri belirli özellikleri ile ön plana çıkarak uygulama geliştiricileri tarafından amaca yönelik programlama dili tercihinin oluşması sağlanmıştır. Örneğin veritabanı uygulaması geliştirecek olan bir uygulama geliştiricinin sistem geliştirme dillerini tercih etmesi ya da yüksek grafik işlemleri için veritabanı uygulama geliştirme dillerini kullanması büyük bir hata olacaktır.

1940'lı yıllarda makine dili ile başlayan bu serüven günümüzde çok ileri programlama dilleri ile devam ediyor olsa da ihtiyaç ve beklentiler bitmemekte ve hala çözüm bekleyen birçok farklı konu bulunmaktadır.

2.2 JStar Genel Özellikleri ve Platform Seçimi

1940'lı yıllarda makine dili ile başlayan yazılım geliştirme süreci farklı evrimler geçirerek günümüze kadar gelmiş ve günümüzde çok ileri programlama dilleri ile devam etmektedir. Fakat bu ileri düzey programlama dilleri de bütün ihtiyaç ve beklentilere cevap vermekte yetersiz kalmaktadır.

Proje konusu olan kaynak bağımlı uygulamaların küme düzeninde çalıştırılması son zamanlarda giderek artan bir önem kazanmaktadır. Günümüz dünyasında geliştirilecek olan bir uygulama için olmazsa olmazlar arasına giren kümeleme, yük dengeleme ve ölçeklenebilirlik kavramları geliştirilmiş olan JStar çatısının temel yapısını teşkil etmektedir.

Günümüzde uygulamaların yalnızca başarılı bir şekilde çalışabiliyor olması ihtiyaçları karşılayamamaktadır. Geliştirilecek olan bir uygulamanın performansı, hızı, ölçeklenebilirliği, kesintisiz hizmet verebilirliği gibi özellikler artık olmazsa olmazlar arasına girmiştir. Bu amaca yönelik olarak geliştirilen JStar çatısı bir uygulamanın birden fazla örneğinin aynı anda aynı ya da farklı makinalarda aktif/aktif kipinde çalışmasına imkan vermektedir. Ayrıca örnekler arasında kaynakların paylaştırılması ile oluşturulan küme içerisinde, yük dağılımı yapılmakta ve bu sayede yük dengeleme gerçekleştirilmektedir. Örneklerden birinin çökmesi ya da kümeden çıkması kümenin ve yük dengelemenin devamlılığı konusunda bir sıkıntı oluşturmamaktadır.

JStar çatısı java programlama dili kullanılarak geliştirilmiş bir çatıdır. Yaygın bir kullanım ağına sahip olan Java programlama dilinin tercih edilmesindeki başlıca etkenler şu şekilde özetlenebilir [3]:

- Açık kaynak olması
- Platform bağımsız olması (Unix/Linux, Windows, Mac OS vb… platformlarda çalışabiliyor olması)
- Yaygın bir kullanıcı komünitesi olması
- Özellikte bellek yönetimi ve güvenlik konularında sunmuş olduğu olanaklar ile uygulama geliştiricilerin rahat geliştirme yapmalarına imkan veriyor olması
- Bir konuda birden fazla alternatif sunuluyor olması. Gelişirme ortamı konusunda; Eclipse Netbeans, jDeveloper vb…, Uygulama sunucusu; Tomcat, WebLogic, JBoss vb… gibi.
- Çok farklı platfromlara hitap ediyor olması. Web (JEE), Mobil (J2ME) vb…
- Gelişmiş çatı desteği sayesinde dilin daha da güçlendirilmiş olması; Spring, Struts, Hibernate, Axis, vb…
- Dünya genelinde destek imkanı

Çizelge 2.2'de günümüzde kullanılar popüler programlama dillerinin kullanım oranları verilmektedir. Buradan da anlaşılacağı gibi Java programlama dilinin tercih edilmesi önemli ve doğru bir karar olmuştur.

Çizelge 2. 2 Popüler programlama dilleri ve kullanım oranları [4]

PROGRAMLAMA DİLİ	KULLANIM ORANI
Java	18.160%
C	16.170%
C++	9.146%
C#	7.539%
PHP	6.508%
Objective-C	5.010%
Python	4.583%
(Visual) Basic	4.496%
Perl	2.231%
Ruby	1.421%
JavaScript	1.394%
Lua	1.102%
Delphi	1.073%
Assembly	1.042%
Lisp	0.953%
Ada	0.747%
Pascal	0.709%
Transact-SQL	0.697%
Scheme	0.580%
RPG (OS/400)	0.503%

2.3 Java Programlama Dili

Java, Sun Microsystems tarafından geliştirilmiş açık kaynak kodlu, platform bağımsız, nesne yönelimli ve yüksek seviyeli bir programlama dilir. İlk sürümü 1995 yılında Sun Microsystems tarafından çıkartılan java programlama dili, herhangi bir donanıma bağlı olmaksızın JVM (Java Virtual machine) adı verilen Java sanal makinası üzerinde çalışmaktadır. Java derleyicileri tarafından üretilen ve byte code adı verilen sınıflar bu JVM üzerinde çalıştırılarak Java'nın en önemli özelliklerinden olan platform bağımsızlığı sağlanmış olur. Şekil 2.1'de bu durum açıklanmaktadır [5].

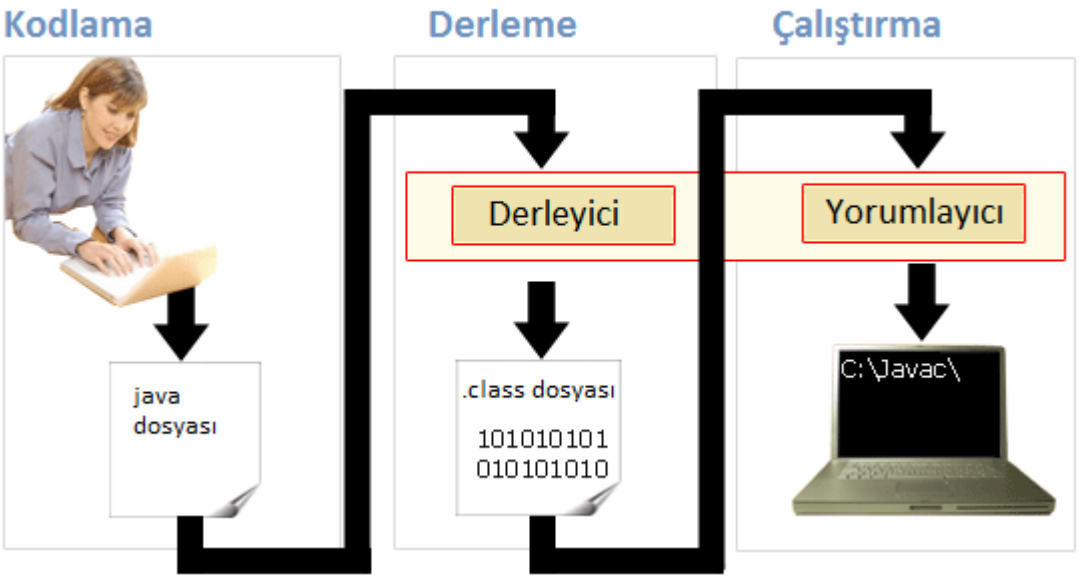
Çizelge 2.3'te Java'nın sürüm listesi kronolojik olarak verilmiştir.

Çizelge 2. 3 Java programlama dilinin sürüm listesi [3]

SÜRÜM	TARİH	AÇIKLAMA
1.0	1996	İlk sürüm
1.1	1997	İçsel sınıflar gibi çeşitli eklentiler yapılmıştır
1.2	1998	Playground olarakta bilinir. Yansıma ve JIT derleyicisi eklenmiştir
1.3	2000	Kestrel olarakta bilinir. Hotspot JVM ilan edildimiştir
1.4	2002	Merlin olarakta bilinir. Pek çok yeni API eklenmiştir
5.0	2004	1.5 olarakta geçmektedir ve Tiger kod adı kullanılmaktadır. Dilin özelliklerinde ciddi değişiklik ve geliştirmeler yapılmıştır
6.0	2006	Mustang olarakta bilinir. Çeşitli geliştirmeler yapılmıştır
7.0	Temmuz 2011	Dolphin olarakta bilinir.
8.0	2012	

Java programlama dili, günvenliği, verimliliği, platform bağımsızlığı (Unix/Linux, Windows, Mac OS vb...), kolay uygulama geliştirilebilirliği (özellikle bellek yönetimi konusunda sundukları), farklı ortamlar için geliştirilmiş olan çözümleri (web, mobil,

masaüstü, oyun programlama) ile ideal teknoloji durumuna gelmiştir. Bugün Java teknolojileri büyük ölçekli uygulama sunucularından, akıllı telefonlara, dizüstü bilgisayarlardan oyun konsollarına, bilimsel amaçlı süper bilgisayarlardan, beyaz eşyalara (buzdolapları, çamaşır makinaları, televizyonlar vb…) kadar çok yaygın bir kullanım ağı bulmuştur [6].

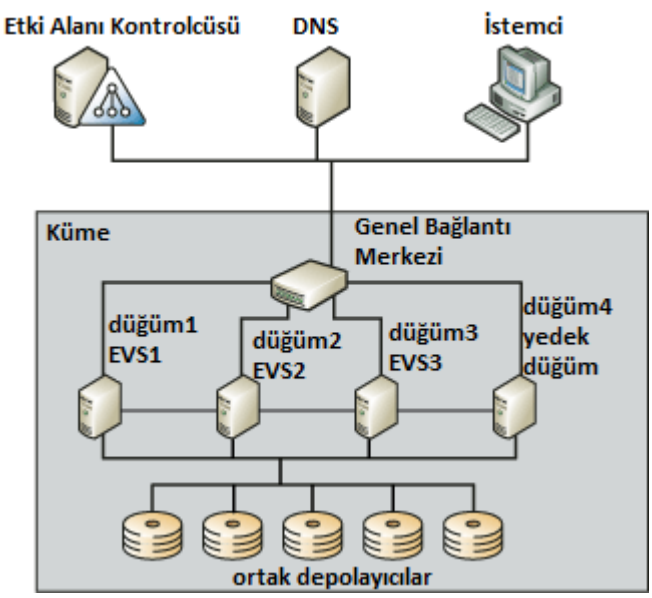


Şekil 2.1 Java yorumlayıcısının çalışması [7]

2.4 Kümeleme

Dışarıdan bakıldığında tek bir makine gibi davranan fakat birden fazla makinadan oluşan ve kullanıcılarına sürekli ve kesintisiz hizmet vermeyi amaçlayan sistemlerdir. kümede bulunan makineler, zorunlu bir koşul olmamakla birlikte genellikle aynı yerel ağ içerisindedirler. Kümeleme sayesinde kesintisiz hizmet ve gelen isteklere daha hızlı cevap verebilecek bir yapı oluşturulmuş olur. Küme içerisindeki bir makinenin çökmesi sistemin çalışmasını aksatmaz, çöken makinenin görevi kümede bulunan diğer makineler ile gerçeklenir. Küme içerisinde bulunan bütün makineler aynı anda aktif durumda olabileceği gibi Şekil 2.2'de gösterildiği üzere bazı düğümler pasif durumda bulunabilir [8].

Kümeleme sistemleri genellikle Gb Ethernet, FDDI vb… gibi yüksek hızlı LAN sistemleri üzerine kurulur.



Şekil 2. 2 Microsoft Exchange sunucusunun 4 düğümlü küme yapısı [9]

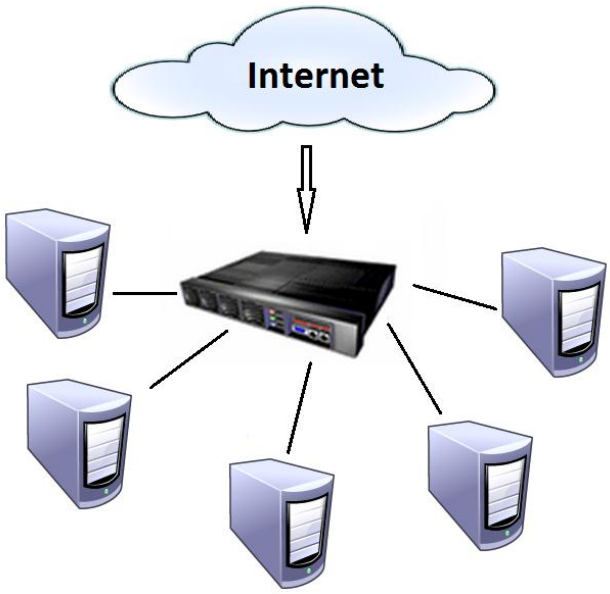
2.5 Yük Dengeleme

Yük dengeleme kısaca bir kümeye gelen isteklerin kümeyi oluşturan düğümler arasında dengeli bir şekilde dağıtılması olarak tanımlanabilir. Web sunucularda bu işi yapmak için yük dengeleyici ismi verilen özel cihazlar kullanılır. Yük dengeleyicilerin yükü dağıtmak için kullandıkları algoritmalar birbirlerinden farklılık gösterebilir.

Yük dengeleme donanım seviyesinde yapılabileceği gibi yazılım seviyesinde de yapılabilir. İdeal bir yük dengeleme algoritmasından yükün küme içerisinde bulunan bütün düğümlere dengeli bir şekilde dağıtması beklenir.

Yük dengeleyiciler durumsuz olabileceği gibi oturum yönetimine ihtiyaç duyan uygulamalarda durumlu da olabilir [10].

Şekil 2.3'te tipik bir yük dengeleyici gösterilmiştir.



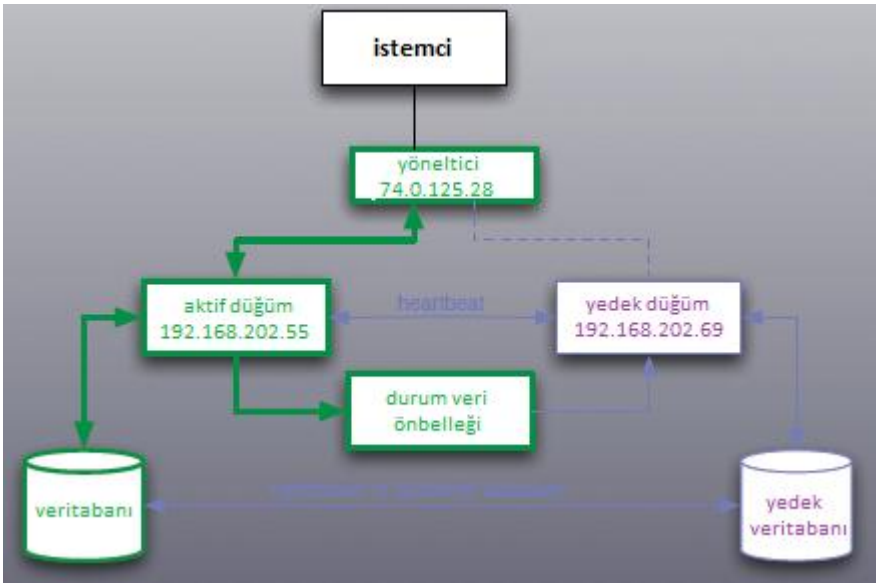
Şekil 2. 3 Dışarıdan gelen istekleri kümeye dağıtan tipik bir yük dengeleyici

2.6 Yüksek Bulunurluk ve Yedeğe Geçme

Daha çok, servisin aktif kalabilirliğini arttırmayı amaçlamaktadır. Bu nedenle atıl düğümler kullanılır. Genellikle her düğümün bir atılı vardır ve pasif durumdadır. Bu atıl düğümler aktif olan düğüm ile aynı verileri içerir ve aktif düğümün çökmesi durumunda pasif durumundan aktif duruma geçerek sistemin devamlılığını veri kaybı olmadan sağlamış olurlar. Normalde İki olan atıl sayısı kritik sistemlerde arttılabilmektedir.

Bu şekilde oluşturulan bir yapı ile sistemin zayıf halkasının çökmesi durumunda ya da bir düğümün çökmesi durumunda bütün sistemin çökmesi anlamı taşıyan bileşenlerden kurtulmuş olunur [11], [12].

Şekil 2.4'te failover durumu gösterilmiştir.



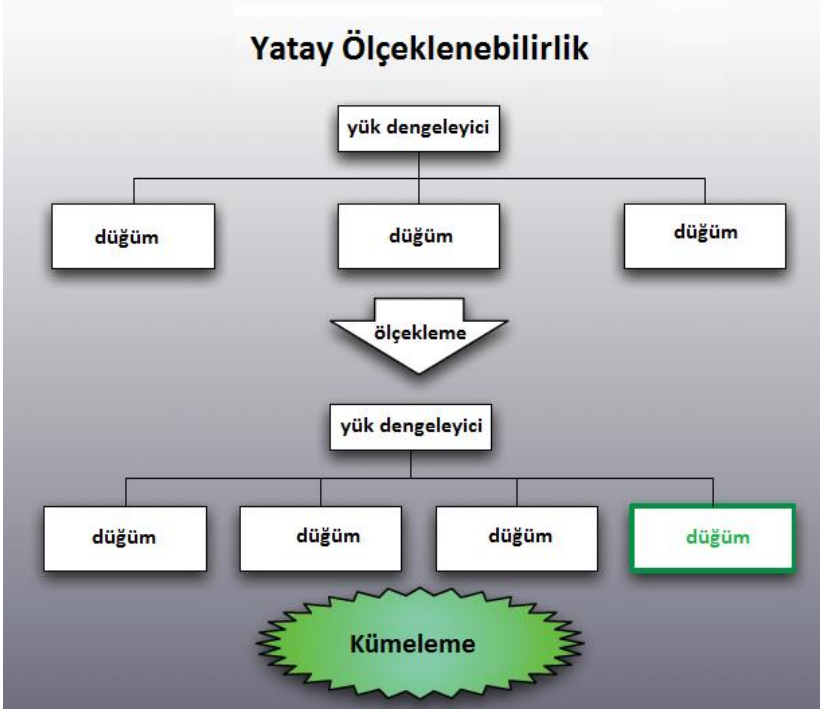
Şekil 2. 4 Yüksek bulunurluk - Yedeğe geçme [12]

2.7 Ölçeklenebilirlik

Ölçeklenebilirlik, sistemin artan taleplerle baş edebilmesi ve bu işi yaparken performansdan feragat etmemesi anlamına gelmektedir. Artan talebe cevap verebilmek için kümeye eklenecek yeni düğümlerin ya da herhangi bir düğümde yapılacak yükseltmelerin sistemin çalışılabilirliğini arttırması ve mevcut sistemin yeni kaynakları efektif bir şekilde kullanabiliyor olması gerekir. Yatay ölçeklenebilirlik ve Dikey ölçeklenebilirlik olmak üzere iki şekli vardır [12], [13].

2.7.1 Yatay Ölçeklenebilirlik

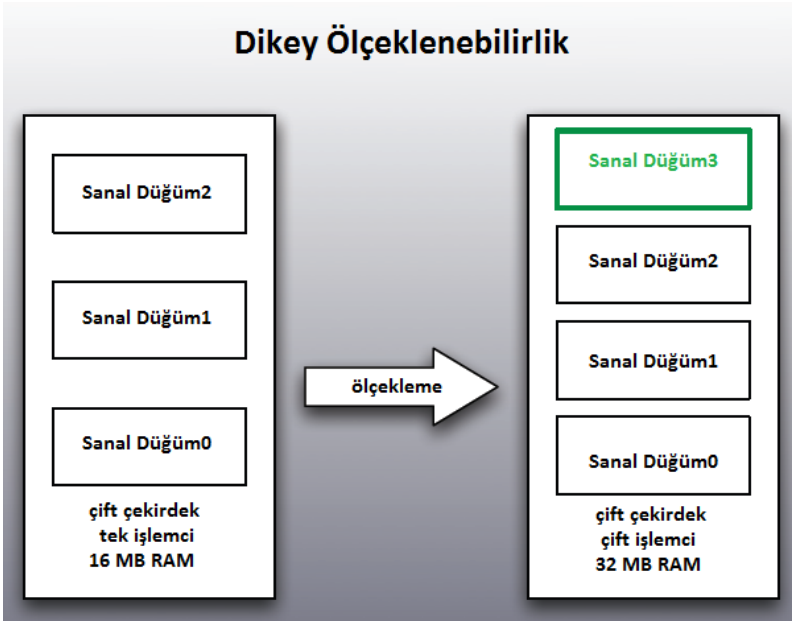
Sisteme mevcutta bulunan düğümler ile aynı özellikte yeni bir düğümün eklenmesi ve yükün yeni eklenen düğüm de hesaba katılarak yeniden dengeli bir şekilde dağıtılmasıdır. Bu durum Şekil 2.5'te gösterilmiştir.



Şekil 2. 5 Yatay ölçeklenebilirlik [12]

2.7.2 Dikey Ölçeklenebilirlik

Sistemede bulunan bir düğümün donanımında yapılan yükseltmeler ile sistemin performansının arttırılmasıdır. Yükseltmeye gidilen donanım parçası ya da parçaları olarak CPU, RAM, storage vb… genellikle performası doğrudan etkileyen birimler seçilir. Dikey ölçeklenebilirlik Şekil 2.6'da gösterilmiştir.



Şekil 2. 6 Dikey ölçeklenebilirlik [12]

2.8 Dağıtımlı Hesaplama

Dağıtık programlama olarak da adlandırılan ve dağıtık bilgi işleme yönteminin sanallaştırılmasını sağlayan çözüm mimarisine kısaca dağılımlı hesaplama denilmektedir.

Buradaki temel amaç dağıtık bilgi işleme ve veri kaynaklarının kullanmakta olduğu işlemci güçleri, ağ kapasiteleri ve depolama kapasiteleri ile tek büyük bir sistem yaratmaktır. Yaratılan bu sistem, tamamen birbirinden bağımsız çalışmakta olan ve birbirine benzemeyen sistemlerin bir araya gelerek oluşturduğu sanal bir işleme gücüdür. Bilgi sistemleri kapasitelerine sonsuz bir kullanım imkanı sunmaktadır.

Dağılımlı Hesaplama, ana yapısı açık standartlardan ve protokollerden oluşmaktadır. Açık dağıtımlı servis mimarisi olarak adlandırılan bu açık mimari heterojendir ve coğrafi dağınık çevrelerin birbirleri arasında haberleşmelerine imkan verir. Dağıtımlı hesaplama ile bir organizasyon içindeki bilgi işleme gücü ve veri kaynakları optimum seviyede kullanılır. Bunun için büyük kapasiteler iş yüklerine ayrılarak (paylaştırılması amacıyla) bütün kaynaklar tarafından işlenir, böylece optimum kullanım sağlanmış olur [14], [15].

Şekil 2. 7 Dağıtımlı hesaplama [15]

Dağıtımlı hesaplama için bilinmesi gereken en önemli üç başlık aşağıdaki gibidir:

- Dağıtımlı hesaplama merkezi olmayan, dağıtık ve sanal bir yapıdır.
- Açık standart ve protokoller ile kurulur.
- Dağıtımlı hesaplama genel iş hedeflerine göre belirlenmiş bir servis kalitesi çerçevesinde çalışmalıdır.

2.8.1 Dağıtımlı Hesaplamaların Faydaları

Dağıtımlı hesaplama paylaştırılabilir bilgi işleme gücü sunar. Günümüzde bilgi işleme ortamları çabuk toparlanan, esnek ve entegre edilebilir olmaya ihtiyaç duymaktadır. Bu nedenle kritik iş süreçleri için dağıtım çözümü uygulanmasının hem iş hem de teknolojik yararları olacaktır.

2.8.1.1 Dağıtımlı Hesaplamanın İş Faydaları

- Zamana bağlı sonuçların üretilmesini hızlandırır
- Kalibrasyonu ve işletim esnekliğini getirir
- İş ihtiyaçları değişkenlerine göre ölçeklenebilir
- Üretimi arttırır
- Ana yatırım maliyetlerini azaltır

2.8.1.2 Dağıtımlı Hesaplamanın Teknolojik Faydaları

- Altyapıda optimizasyon (iş yükü konsolidasyonu vb.) sunar
- Veriye ulaşılabilirliği arttırır
- Kendini toparlayan, yüksek kullanılabilirliği olan bir altyapı sağlar

2.9 Artıklık ve Hataya Dayanıklılık

2.9.1 Artıklık

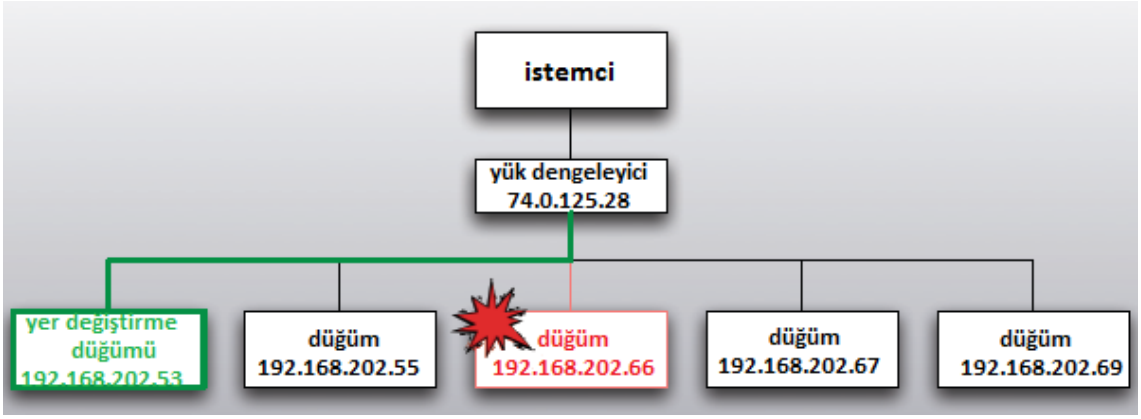
Artıklık, kümede bulunan bir düğümün aynı veriyi tutan farklı bir düğüm ile çoklanmasıdır. Orijinal düğümde bir sorun olması durumunda pasif durumda bulunan aydıl düğüm devreye girer ve kümenin bütününde bir sorun oluşturulmadan küme görevine devam edebilir.

2.9.2 Hataya Dayanıklılık

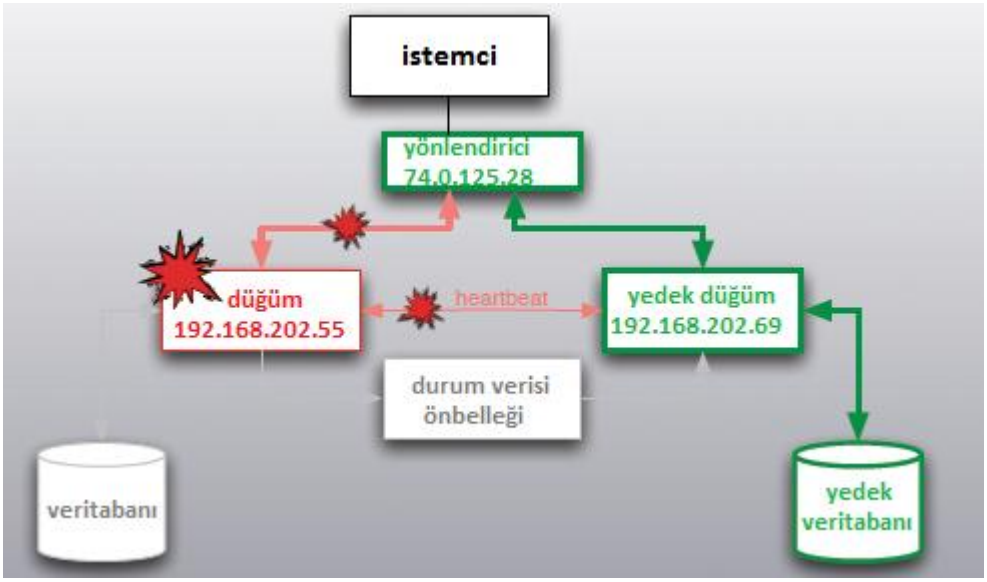
Küme bileşenlerinden birinde oluşan bir sorunun çeşitli yöntemlerle giderilerek kümenin işleyişini aksatmadan devam etmesidir. Böyle bir durum sistem genelinde

belli bir miktar verimlilik kaybı oluştursada sistemin devam ediyor olması büyük bir kazanımdır. Büyük sistemler için kritik öneme sahiptir.

Şekil 2.8'de aktif/aktif çalışan bir sistemde hata dayanıklılığının nasıl gerçeklendiği gösterilmektedir. Yine Şekil 2.9'da de aktif/pasif çalışan bir sistemde hata dayanıklılığının nasıl uygulandığı gösterilmektedir.



Şekil 2. 8 A/A küme hata dayanıklılığı [15]



Şekil 2. 9 A/P küme hata dayanıklılığı [15]

Hata dayanıklılığı 3 farklı yöntem ile gerçeklenir.

Çoğaltma: Gelen istekler ya da işler sistemin aynı özellikli birden fazla örneğine paralel olarak yönlendirilir.

Artıklık: Sistemde aynı özellikli birden fazla örnek bulunur fakat bunlardan yalnızca bir tanesi aktif durumdadır. Aktif düğümlerde bir sorun olması durumunda pasif modda olan örneklerden birine geçilir.

Çeşitleme: Aynı spesifikasyonun birden fazla faklı implementasyonları bulunur ve yedekli sistemlerde olduğu gibi hatanın üzerinden gelmek için özel implementasyon kullanılır.

2.10 JStar'a Neden İhtiyaç Duyulmaktadır?

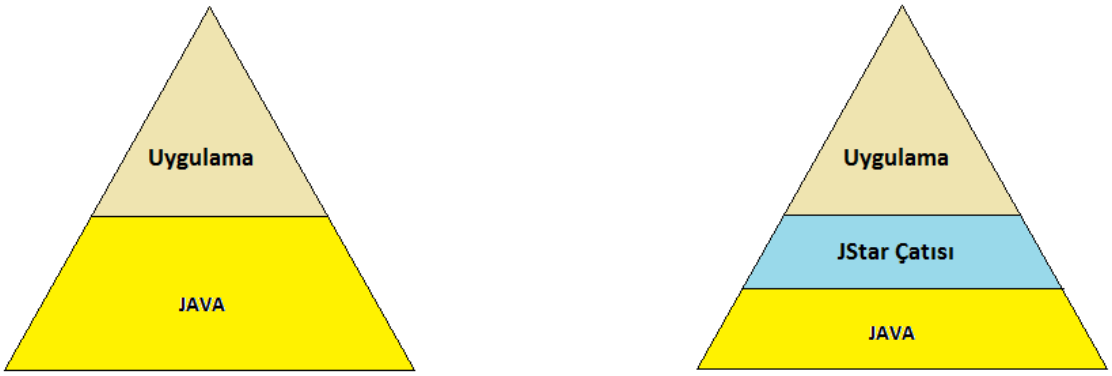
Kümeleme, yük dağılımı, ölçeklenebilirlik gibi kavramlar günümüz dünyasında hemen hemen her uygulamanın ihtiyaç duymakta olduğu kavramlardır. Uygulama sunucular, veritabanı sunucuları gibi önemli ve kritik hizmetler sunan ürünler bu tekniklerin implementasyonunu kendi içlerinde yapmaktadırlar. Fakat uygulama geliştiricilerine bir dosya sistemi erişimde ya da bir veritabanı erişiminde uygulamalarını küme düzeninde çalıştabilmeleri için bir çözüm sunmamaktadırlar. Bu durumda uygulama geliştiricilerin geliştirdikleri her bir uygulama için kendilerine özgü kümeleme çözümleri geliştirmelerine neden olmaktadır. Ve sonuçta hataya açık, kırılgan, kararlı olmayan ve güvenilirliğini ispat edememiş binlerce birbirinden farklı fakat hemen hemen aynı ihtiyaca dönük çözümler oluşmaktadır.

İşte bu çalışma kapsamında bütün bu gereksinimleri (kümeleme, yük dağılımı, ölçeklenebilirlik vb...) karşılamak ve standartlara uygun kararlı bir çözüm üretmek için JStar çatısı geliştirilmiştir. Böylelikle uygulama geliştiricilerin geliştirmekte oldukları aplikasyona daha fazla yoğunlaşmaları, enerjilerini farklı yerlerde tüketmeden uygumalarının iş mantıklarına odaklanmaları hedeflenmiştir.

Şekil 2.10'da stardart bir Java uygulaması ile JStar çatısı kullanılarak geliştirilmiş olan bir java uygulamasının yapısal olarak karşılaştırılması gösterilmiştir. Buradan da rahatlıkla anlaşılacağı gibi JStar çatı programlama dili üzerinde, uygulama katmanı altında bir yerde bulunmaktadır.

Ayrıca bu çalışma kapsamında geliştirilen JStar çatı çözümünün açık kaynak camiasına sunulması planlanmıştır. Bu sayede uygulama geliştiricilerden gelecek eleştiri, öneri ve talepler değerlendirilerek çatı çözümün daha geniş bir kullanım alanı bulması sağlanmış

olacaktır. Ayrıca çatının açık kaynak kodu ile dağıtılması uygulama geliştiricilere de çatı üzerinde geliştirme yapabilme imkanı sunacaktır.



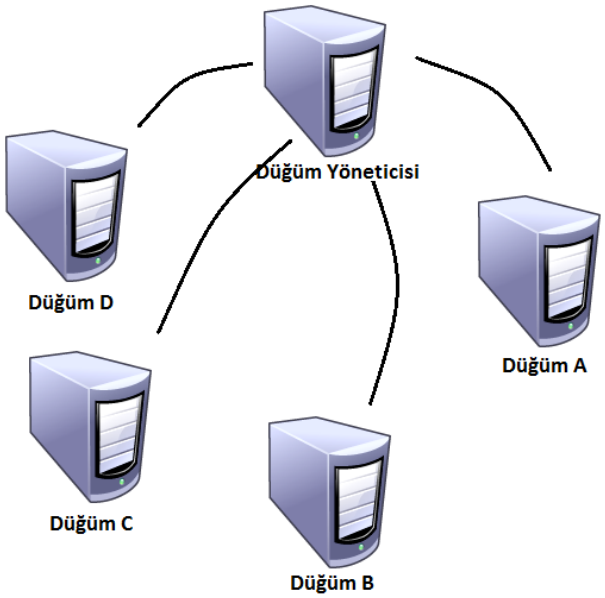
Şekil 2. 10 Standart bir Java uygulaması ve JStar çatısı ile geliştirilen bir Java uygulamasının yapısal karşılaştırılması

BÖLÜM 3

JSTAR JAVA ÇATISI

Kaynak bağımlı bir uygulamaların küme düzeninde çalıştırılabilmesindeki temel problem verinin kaynaktan alınması esnasında ortaya çıkmaktadır. İşlencek olan verinin aynı anda farklı düğümler tarafından da okunması, aynı kaydın birden fazla işleme sokulması anlamına gelecektir. Bir diğer problem de düğümler arasında senkronizasyonun sağlanamaması durumudur. Böyle bir durumda ise bazı kayıtların işlenememesi gibi bir sorun ortaya çıkacaktır [16].

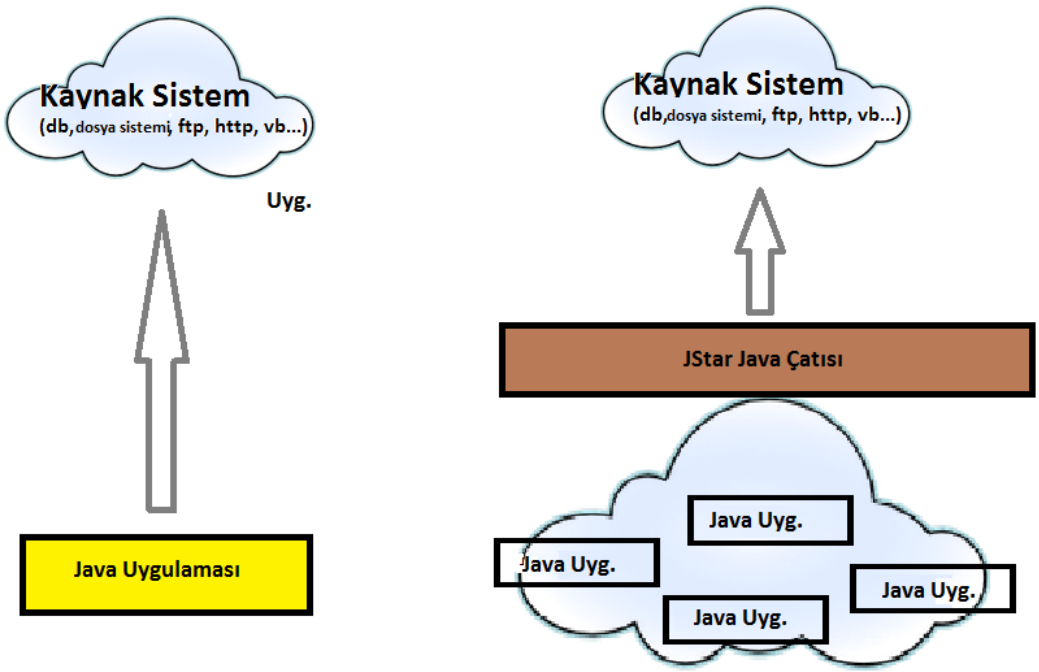
JStar çatısı bu tür problemlerin çözümü için geliştirilmiş bir çatı çözümdür. Bu çatı çözüm sayesinde her bir kaydın mutlak işleneceği ve bir kaydın yalnızca bir kez işleneceğinin garantisi sağlanmıştır. JStar çatısı arka planda oluşturduğu küme sayesinde düğümlere, kaynağa erişim için gerekli yetkiyi ve her bir düğümün hangi kaynakla ilgileneceği bilgisini vermektedir. Bu sayede verinin küme içerisinde güvenli bir paylaşımı gerçeklenmiş olmaktadır.



Şekil 3. 1 JStar küme yapısı

3.1 JStar Küme Genel Yapısı

Şekil 3.2'de JStar çatı çözümü ile geliştirilen bir uygulamanın kaynak erişim yöntemi, standart bir Java uygulamasının kaynak erişim yöntemi ile karşılaştırılmaktadır. Günümüz uygulama geliştirme ihtiyaçları dikkate alındığı zaman JStar çatı çözümünün sağlamış olduğu imkanlar daha iyi anlaşılacaktır.



Şekil 3. 2 Standart bir java uygulaması ve JStar çatısı ile geliştirilen bir java uygulamasının kaynak erişim yapıları

JStar çatı çözümünü kullanarak çalıştırılacak her bir uygulama iki farklı başlatılma yönteminden birini kullanmak zorundadır. Bunlar; JStar düğüm yöneticisi ve JStar düğümüdür.

3.2 JStar Küme Bileşenleri

JStar kümesinde bulunan her bir düğüm yaptığı iş itibariyle ya küme düğümü olarak ya da küme düğüm yöneticisi olarak adlandırılır. Bir JStar kümesi içerisinde yalnıza bir adet küme düğüm yöneticisi bulunurken küme düğüm sayısı tamamen değişken olup bir ya da birden fazla olabileceği gibi hiç olmadığı durumlarda oluşabilir. Aslında kümede bulunan düğüm yöneticisi aynı zamanda bir küme düğümüdür ve bir düğümün bütün görevlerini gerçekleştirmektedir. Bu görevler dışında kümenin yönetsel görevlerini de üstlenmiştir. Benzer şekilde kümede bulunan her bir düğüm, küme düğüm yöneticisi görevlerini yapabilecek kapasitede olduğu halde bu özelliklerini kullanmaz. Ta ki küme tarafından kendisine küme düğüm yöneticiliği görevi verilene kadar.

3.2.1 JStar Küme Düğüm Yöneticisi

Uygulama ilk başlatılırken ilgili düğüme bir düğüm yöneticisi olduğu bilgisinin aktarılması ile gerçekleştirilir ve ayağa kaldırılan örnek, bir düğüm gibi çalışmakla birlikte aynı zamanda düğüm yöneticilik görevini de üstlenmiş olur. Bu işlemin gerçekleştirilebilmesi için başlangıç kısmında tekil desen ile gerçeklenmiş olan tr.edu.yildiz.jstar.shell.facade.CloudNotifier sınıfının getInstance metodunun Şekil 3.3' de gösterilen parametreler (aslında yalnızca listenPort bilgisine ihtiyaç duyulmaktadır, kendisi düğüm yöneticisi olacağı için herhangi bir düğüm yöneticisi bilgisi verilmez) ile başlatılması gerekmektedir.

```
CloudNotifier.getInstance(listenPort, null, null);
```

Şekil 3. 3 Bir düğümün düğüm yöneticisi olarak başlatılması

Burada listenPort olarak verilen parametre düğüm yöneticisinin diğer düğümler ile iletişim kuracağı TCP kapı bilgisini belirtmektedir.

Bu şekilde başlatılacak olan bir düğüm kümedeki düğüm yöneticiliği rolünü de üstlenmiş olacaktır ve düğümlerden gelecek olan isteklere küme şartlarına uygun cevaplar vererek kümenin düzenli bir şekilde kesintisiz çalışmasını sağlayacaktır.

3.2.2 JStar Küme Düğümü

Bir diğer başlatılma yöntemi ise düğümün ekstradan bir görevi olmaksızın bir düğüm yöneticisine bağlı olarak çalıştırılmasıdır. Bu işlemin de gerçekleştirilebilmesi için başlangıç kısmında tekil desen ile gerçeklenmiş olan tr.edu.yildiz.jstar.shell.facade.CloudNotifier sınıfının getInstance metodunun Şekil 3.4' de gösterildiği gibi listenPort, managerIP ve managerPort bilgileri ile başlatılması gerekmektedir.

```
CloudNotifier.getInstance(listenPort, managerIP, managerPort);
```

Şekil 3. 4 Bir düğümün küme düğümü olarak başlatılması

Burada geçen parametreler ve açıklamaları şu şekildedir;

listenPort: Düğümün, düğüm yöneticisi ve diğer düğümler ile iletişim sağlayabileceği TCP kapı değerini göstermektedir.

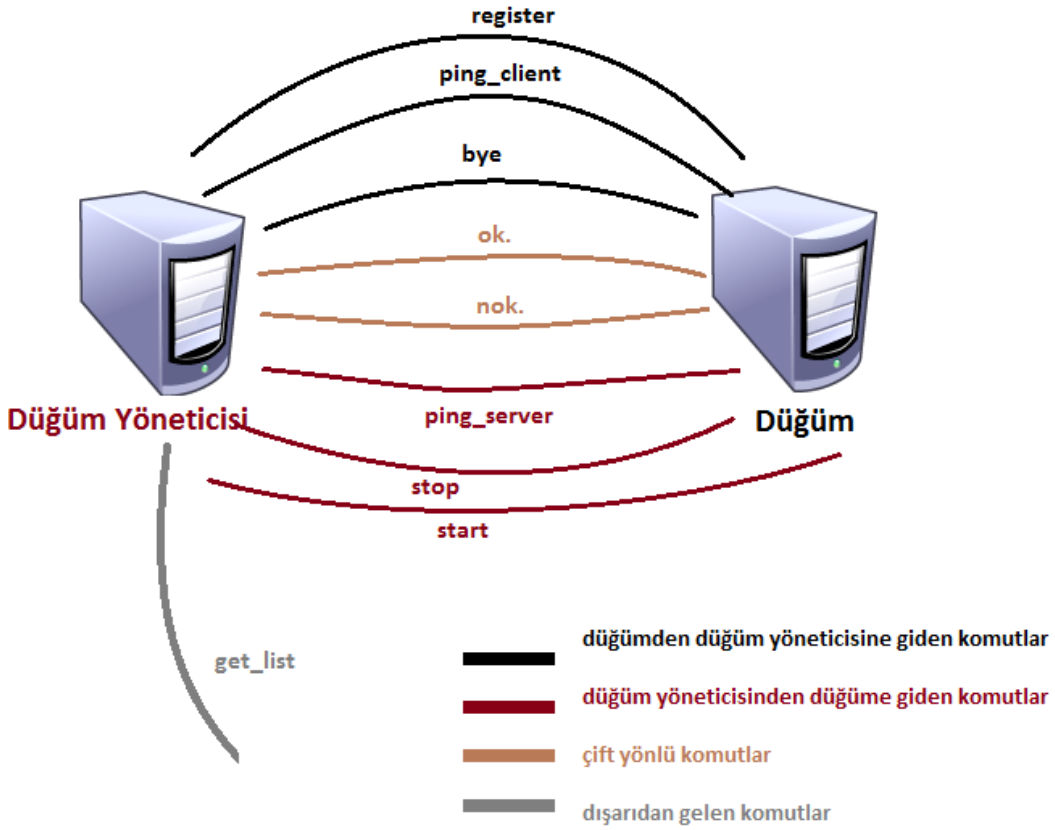
managerIP: Kümede bulunan yöneticinin TCP/IP değerini ifade etmektedir. Bu sayede düğüm, küme düğüm yöneticisi ile gerekli iletişimi kurabilecektir.

managerPort: Kümede bulunan yöneticinin TCP kapı değerini ifade etmektedir. Bu sayede düğüm, küme düğüm yöneticisi ile gerekli iletişimi kurabilecektir.

3.3 JStar Kümesi Haberleşme Yöntemleri

JStar Java çatısında haberleşme p2p (eşler arası) TCP protokolü kullanılarak gerçekleştirilmiştir. Bir çok kurumsal firmanın anahtarları güvenlik kaygıları dolayısı ile toplu yayın mesajlarını bloke edecek şekilde konfigüre edildiğinden bu çalışmada toplu yayınlama protokolünden kaçınılmıştır.

Şekil 3.5'te JStar Java çatısı kapsamında kullanılan bütün mesajlar ve bu mesajların yönleri gösterilmiştir.



Şekil 3. 5 JStar Java çatısının mesajlaşma yapısı

JStar Java çatısı, TCP protokolü üzerinde kendi mesaj formatını kullanmaktadır. Bu mesaj formatının yapası Şekil 3.6'da gösterildiği gibidir.

[[cmd][par1][par2][par3]...[parN]]

Şekil 3. 6 JStar Java çatısının kullanmış olduğu mesaj formatı

İlk parametere komut tipini belirtmektedir. Devamında gelen parametreler ise ilgili komuta bağlı olarak değişmektedir ve komut ile kullanılacak anlamlı veriyi içermektedir.

JStar çatısı kapsamında tanımlanmış bütün komutlar ve bunların açıklamaları aşağıdaki gibidir:

register: Yeni bir düğüm kümeye dahil olmak istediğinde küme düğüm yöneticisine göndermesi gereken komut "register" komutudur. Komutun tam yapısı Şekil 3.7'de gösterildiği gibidir. Burada birinci parametre komutun kendisini ifade ederken ikinci

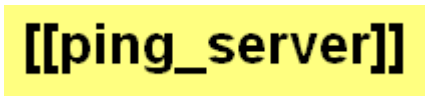
parametre de düğüme hangi kapı üzerinden erişilebileceği bilgisini içermektedir. (IP bilgisi zaten bağlantıdan alınabildiği için tekrardan gönderilmesine gerek duyulmamaktadır.)

Şekil 3. 7 “register” komutu yapısı

ping_client: Her bir düğümün belirli aralıklarla düğüm yöneticisine göndermesi gereken bir komuttur. Bu sayede ilgili düğümün ayakta olduğu ve veri işleme görevini gerçekleştirdiği bilgisi düğüm yöneticisi tarafından izlenebilmektedir. Komutun yapısı Şekil 3.8’de gösterilmiştir. Birinci parametre komutun kendisini ifade ederken ikinci parametre de düğümü kümede tanımlayan Node ID değerini içermektedir.

Şekil 3. 8 “ping_client” komutu yapısı

ping_server: Belirtilen zamanaşımı süresi boyunca herhangi bir düğümden “ping_client” komutu alınmaz ise düğüm yöneticisi tarafında ilgili düğüme gönderilen komuttur. Komutun yapısı Şekil 3.9’da gösterildiği gibidir. Bu komut düğüm yöneticisi tarafından belirli bir zamanaşımı süresi boyunca yine belirli aralıklarla ilgili düğüme gönderilir. Belirtilen zamanaşımı süresi boyunca ilgili düğümden gerekli cevap alınamaz ise düğümün kümeden ayrıldığı kararına varılarak küme yapılandırılmasına gidilir.



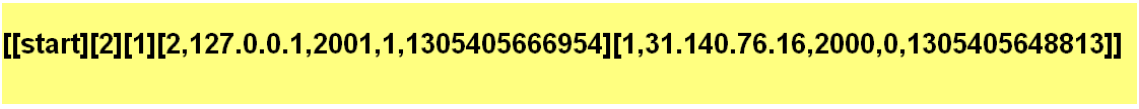
Şekil 3. 9 “ping_server” komutu yapısı

stop: Yeni bir küme oluşturulması durumunda düğüm yöneticisi tarafında bütün düğümlere gönderilen ve yeni bir küme yapılandırılmasına gidilecek olmasından dolayı, düğümlerin veri işleme durumundan çıkıp bekleme durumuna girmeleri gerektiğini ifade eden komuttur. Bu şekilde yeni küme kurulana kadar düğümlerin kayıt işlemesinin önüne çekilmiş olur. Komutun yapısı Şekil 3.10 gösterilmiştir.

Şekil 3. 10 “stop” komutu yapısı

start: Düğüm yöneticisinden kümedeki bütün düğümlere gönderilen ve yeni kümenin kurulduğunu ifade eden komuttur. Bu komut ile kümenin bütün yapısı, kümede hangi düğümlerin bulunduğu, bu düğümlerin hangi kapılardan hizmet verdiği ve her bir düğümün hangi ID'ye sahip olduğu bilgisi küme düğümlerine iletilir. Bu komutu alan her bir düğüm kendi parametreleri ile bekleme durumudan veri işleme durumuna geçerek veri işlemeye başlar. Komutun yapısı Şekil 3.11’de verilmiştir. Birinci parametre komutun kendisini ifade etmektedir. İkinci parametre olan modeCount kümede bulunan toplam düğüm sayısını (düğüm yöneticisi de dahil olmak üzere) ifade eder. Üçüncü parametre olan modeID, komutun gönderildiği düğümün veri erişiminde kullanacağı ID'yi ifade eder. Node1, Node2,..,NodeN şeklinde devam eden parametreler ise kümedeki her bir düğümü ifade eden tanımlayıcı bilgiler içerir. Formatı Şekil 3.13’te verilmiştir. NodeID düğümün kümedi ID'sini ifade etmektedir. nodeIP düğümün IP bilgini ifade eder, nodePort değeri düğümün kapı bilgisini belirtir. NodeType düğümün yalnızca bir düğüm mu yoksa aynı zaman da düğüm yöneticisi mi olduğu bilgisini gösterir. LastUpdateTime düğümün göndermiş olduğu son komutun zaman değerini gösterir. Şekil 3.12’de “start” komutunun gerçek bir sistemden alınmış bir örneği verilmiştir.

Şekil 3. 11 “start” komutu yapısı



Şekil 3. 12 Örnek “start” komutu

Şekil 3. 13 “start” komutunda kullanılan düğüm bilgi parametresi

ok.: Bir komuta verilen iki farklı cevaptan başarılı olanı ifade eder. Komut formatı Şekil 3.14'te gösterilmiştir.

Şekil 3. 14 "ok." komutu yapısı

nok.: Bir komuta verilen iki farklı cevaptan başarısız olanı ifade eder. Komut formatı Şekil 3.15'te gösterilmiştir.

Şekil 3. 15 "nok." komutu yapısı

bye: Kümeyi tamamen terketmek için kullanılan bir komuttur. Komut formatı Şekil 3.16'da gösterilmiştir.

Şekil 3. 16 "bye" komutu yapısı

get_list: Küme hakkında bilgi almak için dışarıdan çalıştırılabilecek bir komuttur. Komut yapısı Şekil 3.17'da gösterilmiştir.

Şekil 3. 17 "get_list" komutu

Dışarıdan uç birim kullanılarak küme düğüm yöneticisine bağlanılabilir ve bu komut sayesinde küme hakkında bilgi alınabilir. Şekil 3.18'de örnek bir kullanımı gösterilmiştir.

```
yasin-kayas-MacBook:~ ysnky$ telnet 127.0.0.1 2000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[[get_list]]

-----------------------------------NODE-----------------------------------
id:2
ip:127.0.0.1
port:2001
type:slave
mode:1/3
lastupdate:1305408327465
--------------------------------------------------------------------------------

-----------------------------------NODE-----------------------------------
id:1
ip:31.140.76.16
port:2000
type:master
mode:0/3
lastupdate:1305405648813
--------------------------------------------------------------------------------

-----------------------------------NODE-----------------------------------
id:4
ip:127.0.0.1
port:2002
type:slave
mode:2/3
lastupdate:1305408333274
--------------------------------------------------------------------------------
```

Şekil 3. 18 “get_list” komutu sonucu alınan küme bilgisi

3.4 JStar Kümesi Düğüm Statüleri

JStar Java kümesi içerisinde bulunan bir düğüm 2 farklı statüden birinde bulunabilir. Bu statüler aynı zamanda düğümün görev ve kimlik tanımını da ifade etmekte olup açıklamaları şu şekildedir:

NODE_TYPE_MASTER: Küme düğüm yöneticilik görevini belirtir ve düğümün yaşam döngüsü boyunca geçerlidir. Bu görevden farklı bir göreve geçiş mümkün değildir.

NODE_TYPE_SLAVE: Küme düğüm görevini belirtir. Düğüm yöneticisinin çökmesi durumunda yeni düğüm yöneticisi olarak atanır ve bu durumda NODE_TYPE_MASTER görevine geçiş yapar. NODE_TYPE_MASTER görevine geçiş yapan bir düğüm yaşam

döngüsü boyunca bu görevi ifa etmek zorunda olup tekrar eksi göreve geçişi mümkün değildir.

3.5 JStar Kümesi Düğüm Durumları

JStar Java kümesi içerisinde bulunan bir düğüm 3 farklı durumdan birinde bulunabilir. Bu durumlar ve açıklamaları şu şekildedir:

STATE_INIT: Bir düğümün başlangıç durumunu ifade eden durumdur. Her bir düğüm kümeye dahil olana kadar bu durumda bulunur.

STATE_WAIT: Kümeye dahil olmuş bir düğümün yeni bir küme kurulmadan önce bulunmuş olduğu durumdur. Bu duruma düğüm yöneticisinden alınan “stop” komutu ile geçilir ve yine düğüm yöneticisinin göndereceği “start” komutu ile bu durumdan çıkılır.

STATE_GO: Kümede bulunan düğümün veri işleme yapabileceği tek durumdur. “start” komutu ile bu duruma geçilir ve “stop” komutu ile bu durumdan çıkılır.

Bunların dışında bir de düğüm yöneticisinin kümeyi yapılandırırken kullanmakta olduğu ve düğümlerden almış olduğu bilgilere göre belirlediği içsel durumlar vardır. Bu içsel durumlar ve açıklamaları şu şekildedir:

INTERNAL_STATE_INIT: Düğüm yöneticisi tarafından yeni bir küme kurulmadan önce bütün düğümlerin durdurulması ve yeni küme oluşturulduktan sonra bütün düğümlerin başlatılması gerekmektedir. Düğümlerin durdurulması ve başlatılması işlemlerinden önce düğüm yöneticisi elinde bulundurduğu küme kayıt tablosundaki bütün düğümlerin içsel durumunu INTERNAL_STATE_INIT olarak günceller. Böylelikle düğümler üzerinde yeni bir işlem başlatılmış olur.

INTERNAL_STATE_STOP: Kümeki düğümleri durdurma işlemi öncesinde bütün düğümlerin içsel durumları INTERNAL_STATE_INIT olarak güncellenir. Bir sonraki adımda ise her bir düğüme “stop” komutu gönderilir ve düğümlerden gelen “ok.” cevabı ile düğüm yöneticisinde bulunan küme kayıt tablosundaki ilgili düğümlerin içsel durumu INTERNAL_STATE_STOP olarak güncellenir. Böylelikle hangi düğümlerin durdurulduğu bilgisi düğüm yöneticisi tarafından kolaylıkla takip edilmiş olur.

INTERNAL_STATE_GO: Kümedeki düğümlerin başlatılması işlemi öncesinde bütün düğümlerin içsel durumları INTERNAL_STATE_INIT olarak güncellenir. Bir sonraki adımda ise her bir düğüme “start” komutu gönderilir ve düğümlerin “ok.” komutu ile cevap vermesi durumunda düğüm yöneticisinde bulunan küme kayıt tablosunda düğümlerin içsel durumları INTERNAL_STATE_GO olarak güncellenir. Böylelikle hangi düğümlerin başlatıldığı bilgisi düğüm yöneticisi tarafından kolaylıkla takip edilebilmiş olur.

3.6 JStar Kümesi Bileşenlerinin Görevleri

JStar çatısı ile oluşturulan küme içerisinde iki farklı düğüm vardır. Bunlar; küme düğümü ve küme düğüm yöneticisidir. Aslında her bir düğüm, düğüm yöneticisi özelliklerini de bünyesinde barındırır fakat bu özellikler pasif durumdadır. Düğüm yöneticisinin çökmesi durumunda yeni düğüm yöneticisi olarak tayin edilen düğüm, pasif durumudaki yönetsel fonksiyonlarını aktif ederek kümeden gelen mesajlara ilgili cevaplar üreterek kümeyi yönetir. Düğüm yöneticisi görevine getirilen bir düğüm aynı zamanda normal bir düğüm gibi küme içerisindeki görevlerini icra etmeye devam eder. Ve düğüm yöneticiliğine terfi etmiş olan bir düğümün tekrardan yöneticilik görevini bırakarak küme düğümüne dönmesi mümkün değildir.

3.6.1 JStar Kümesi Düğüm Yöneticisinin Görevleri

JStar kümesinde bulunan her bir düğüm aslında düğüm yöneticiliği görevini yapabilecek kapasitedir fakat yöneticilik görevi yalnızca bir düğüm tarafından gerçeklenir. Düğüm yöneticilik görevi ya uygulama ilk ayağa kaldırılırken verilir ya da kümenin düğüm yöneticisinin düşmesi durumunda kümede bulunan diğer düğümler tarafından ortaklaşa kararlaştırılarak bir düğüme verilir.

Şekil 3.19’da bir uygulamanın düğüm yöneticiliği rolünde çalıştırılması gösterilmiştir.

```
ysnky$ java -cp .:mysql-connector-java-5.1.14-bin.jar tr.edu.yildiz.jstar.test.JStarTest 2000
```

Şekil 3. 19 Test uygulamasının düğüm yönetisi olarak çalıştırılması

Uygulama ilk başlatılırken düğüm yöneticisi olarak belirtilmiş ya da küme tarafından düğüm yöneticisi olarak seçilmiş bir düğüm, kümenin sağlıklı çalışmasından ve devamlılığından sorumludur. Bir düğüm yöneticisinin temel görevleri şu şekilde maddelendirilebilir:

- Kümenin oluşturulması
- Yeni bir düğümün kümeye eklenmesi
- Kümeden ayrılan ya da çöken bir düğümün belirlenerek kümeden çıkartılması
- Kümeye gelecek olan komutları değerlendirip gerekli cevapları en kısa sürede oluşturabilmesi

Düğüm yöneticisi olarak başlatılan bir düğüm kümedeki düğümlerden gelecek komutlara cevap verebilecek kabiliyete sahiptir. Kümeden gelen komutlar duruma göre senkron ya da asenkron olmak üzere iki farklı şekilde işleme tabi tutulmaktadır. Kümenin tamamını ilgilendirmeyecek tarzdaki komutlar (“ping_client” gibi) senkron bir şekilde anında işlenip düğüme ilgili cevap verilirken bütün kümeyi ilgilendiren komutlar (“register”, “bye”, vb...) düğüm yöneticisi tarafında bir kuyruğa atılır ve farklı işleyici ya da işleyiciler tarafında asenkron olarak işlenir.

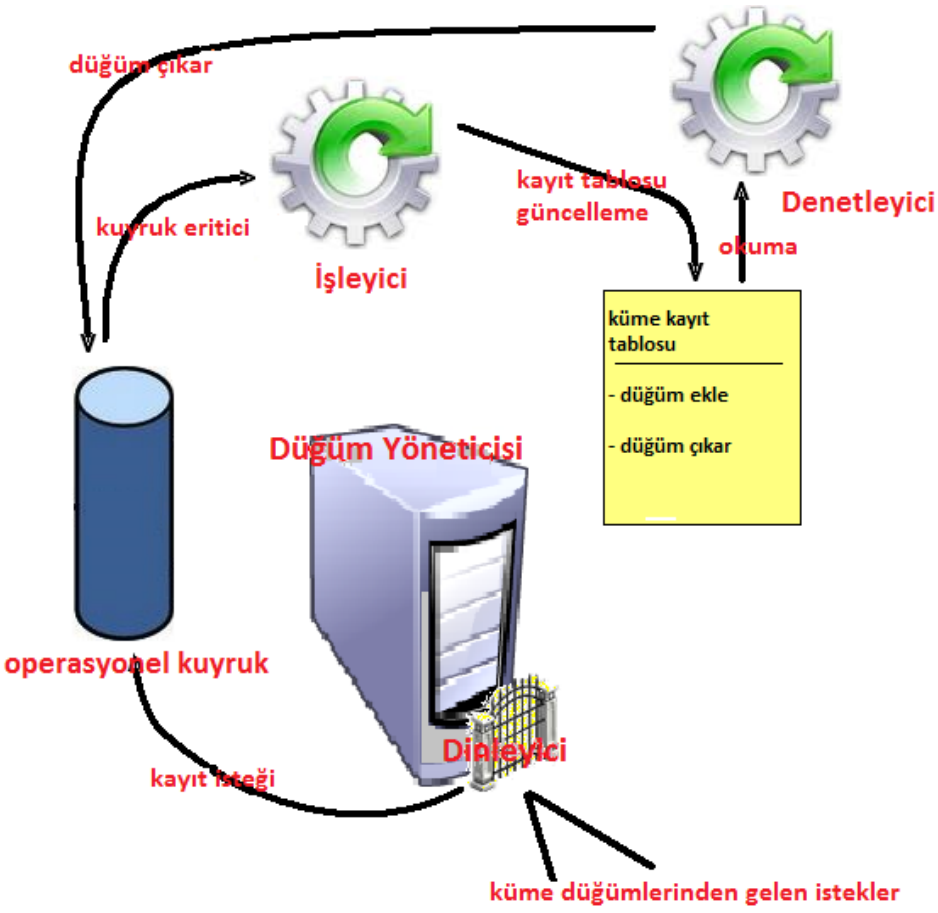
Düğüm yöneticisi bütün küme düğümleri hakkında gerekli bilgileri tuttuğu bir içsel hafıza kayıt tablosu ile asenkron komutları tutuğu bir kuyruğa sahiptir. Bu kayıt tablosu ve kuyruk sayesinde küme hakkında gerekli bilgilere ve işletilmesi gerekli olan komutlara sahip olmuş olur.

Düğüm yöneticisinin kümeyi yönetmesinde kullanılan temel içsel bileşenler aşağıdaki gibidir:

tr.edu.yildiz.jstar.core.network.NodeAcceptor: Kümede bulunan düğümlerden gelen isteklere cevap üreten temel bileşendir. Gelen komutlar tiplerine göre senkron ya da asenkron olarak cevaplandılır.

tr.edu.yildiz.jstar.core.engines.ControllerEngine: Bu birim sayesinde küme belirli aralıklarla monitör edilerek zamanaşımı sürelerinde “ping_client” komutu üretmeyen düğümler tespit edilir ve bu düğümlerin kümeden çıkartılması için asenkron komut üretilir.

tr.edu.yildiz.jstar.core.engines.ProcessorEngine: ControllerEngine birimi tarafından kümeden çıkartılacak olan düğümler veya "NodeAcceptor" tarafından kümeye dahil edilecek ya da kümeden çıkartılacak olan düğümlere ait bigiler içsel bir kuyruğa atılır. Bu kuyrukta biriken komutlar ProcessorEngine aracılığı ile belirli aralıklarla işlenerek küme yeniden yapılandırılır ve sonuçta oluşan yeni küme yapısı yine bu birim aracılığı ile bütün küme düğümlerine bildirilir. ProcessorEngine biriminin işlemiş olduğu komutlar asenkron komutlardır.



Şekil 3. 20 JStar küme düğüm yönetici yapısı

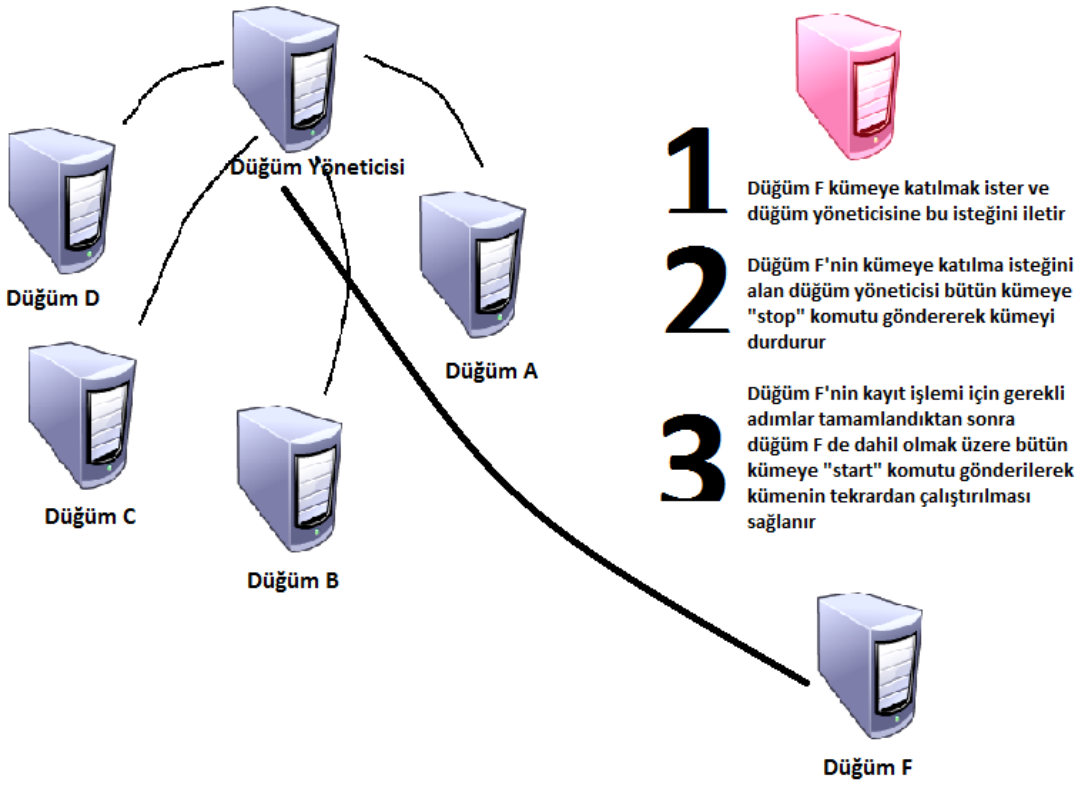
3.6.1.1 Kümeye Yeni Bir Düğümün Eklenmesi

Düğüm yöneticisine "register" komutu ile gelen istekler kümeye yeni bir düğümün eklemesini tetikler. NodeAcceptor aracılığı ile alınan "register" komutu kuyruğa OPERATION_ADD olarak aktarılır ve bir sonraki döngüde işleme alınması sağlanır. ProcessorEngine, OPERATION_ADD şekinde bir komut ile karşılaştığında öncelikli olarak bütün düğümlere "stop" komutu gönderir ve düğümlerin buna "ok." komutu ile cevap

vermesi beklenir. Belirtilen zamanaşımı süresinde cevap vermeyen düğümler çökmüş olarak değerlendirilerek kümeden çıkartılır ve “stop” işlemi baştan tekrarlanır. Bütün düğümlerden “ok.” cevabı alındıktan sonra kümenin durdurulduğundan emin olunarak yeni küme oluşturulması işlemine geçilir.

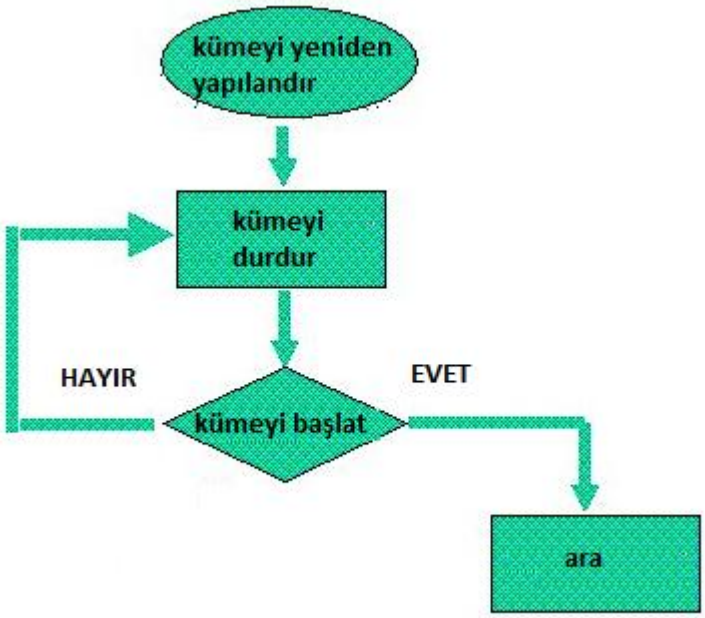
“stop” komutuna “ok.” cevabı dönen her bir düğüm STATE_WAIT durumuna geçerek veri işleme sürecini durdurur.

Bu aşamada aktif düğüm sayısından yararlanılarak modeCount değeri bulunur. modeId değeri ise düğüm yöneticisi 0 değerini alacak şekilde ve sıra ile her bir düğüm için artan değerler atanarak belirlenir ve yeniden hesaplanan bu bilgiler düğümlere “start” komutu eşliğinde gönderilir. Yeni gelen düğüm için bir nodeID gereksinimi duyulduğundan tekil dizayn deseni kullanılarak üretilen ve artarak giden bir değer olup küme yaşam döngüsü boyunca geçerli olan nodeID değeri yeni düğüm için nodeID olarak belirlenir ve yeni düğüme “start” komutu ile gönderilir. “start” komutunu alan her bir düğüm aynı zamanda kendi nodeID'sini (düğüm yöneticisinin düşmesi durumunda kritik öneme sahip olan ID), kümedeki düğüm sayısını veren modeCount değeri ve veri erişiminde kullanabileceği modeID değerlerini almış olur. Bunların yanısıra kümede kimlerin bulunduğunu belirten bir küme listesini de almış olur. Ve bu bilgiler doğrultusunda çalışmasına devam eder. Kümeye yeni bir düğüm eklenmesi durumu Şekil 3.21’de gösterilmiştir.



Şekil 3. 21 JStar kümesine yeni bir düğümün eklenmesi

Bu işlemler esnasında “start” komutuna “ok.” dönmeyen bir düğüm olması durumunda “start” işlemi yarıda kesilerek en başa dönülür. Yani önce bütün küme tekrardan “stop” ettirilir ve yine tekrardan “start” edilir. Bu işlem “start” komutuna bütün düğümlerin “ok.” dönmesine kadar devam ettirilir. Şekil 3.22’de JStar kümesinin tekrardan çalıştırılması için kullanılan yöntem verilmiştir.



Şekil 3. 22 JStar kümesinin yeniden yapılandırılmasında izlenen yöntem

"start" komutuna "ok." cevabını veren her bir düğüm STATE_WAIT durumundan STATE_GO durumuna geçerek veri işleme sürecine kaldığı yerden yeni düğüm değerleri ile devam eder.

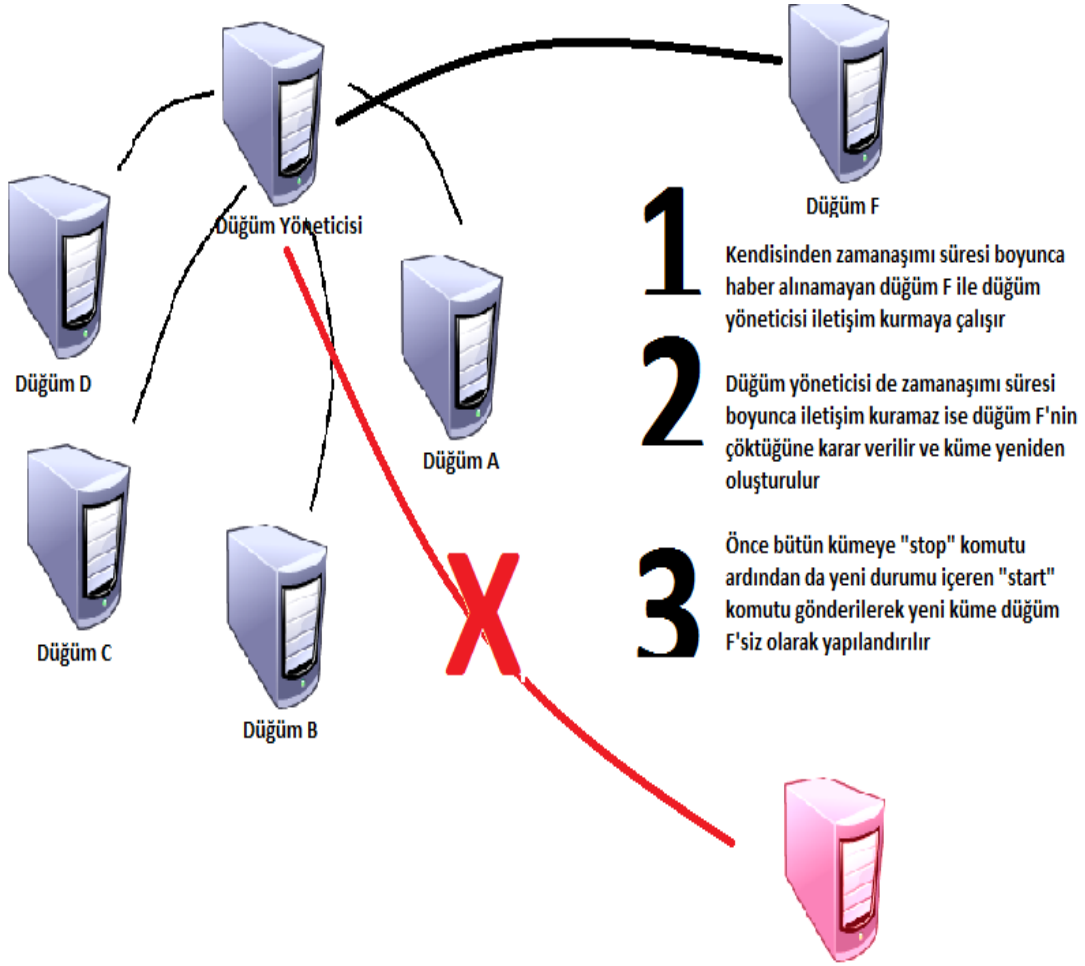
3.6.1.2 Kümeden Bir Düğümünden Düşmesi

Düğüm yöneticisine "bye" komutu ile gelen istekler kümeden ilgili düğümün çıkartılması sürecini tetikler. NodeAcceptor aracılığı ile alınan "bye" komutu kuyruğa OPERATION_DEL olarak aktarılır ve bir sonraki döngüde işlenmesi sağlanır.

Bu işlem belirtilen "client_ping_timeout" süresinde iletişim kuramamış olan düğümler için otomatik olarak controllerEngine tarafından yapılır. "client_ping_timeout" süresi boyunca iletişim kuramamış düğüm için düğüm yöneticisi "server_ping_timeout" süresi boyunca "server_ping" göndermeye çalışır. Eğer ilgili düğüm bu süre boyunca da cevap veremez ise düğüm, operasyonel kuyruğa OPERATION_DEL olarak aktarılır ve bir sonraki döngüde ilgili komutun işlenmesi sağlanır.

ProcessorEngine, OPERATION_DEL şekinde bir komut alındığında öncelikli olarak ilgi düğümü küme kayıt tablosundan çıkartılır. Yeni durum için küme yapılandırılmasına gidilir. Bu işlem için öncelikle bütün düğümlere "stop" komutu geçilir ve düğümlerin bu komuta "ok." komutu ile cevap vermesi beklenir. Belirtilen zamanaşımı süresinde

cevap veremeyen düğümler çökmüş olarak değerlendirilerek kümeden çıkartılır. Ve durdurma işlemi baştan tekrarlanır. Bütün düğümlerden “ok.” cevabı alındıktan sonra kümenin durduğundan emin olunarak yeni küme oluştrulması işlemine geçilir ve yeni şartlara göre oluşan küme ve düğüm değerleri düğümlere “start” komutu verilerek aktarılır ve böylelikle yeni kümenin oluşturulması işlemi tamamlanmış olur. Bu durum Şekil 3.23’te gösterilmiştir.



Şekil 3. 23 JStar kümesinde bir düğümün düşmesi

3.6.2 JStar Küme Düğümünün Görevleri

JStar kümesinde düğüm yöneticisi ip ve kapı değerleri verilerek ayağa kaldırılan her bir düğüm, küme düğümü olarak adlandırılır ve düğüm yöneticisi için gerekli olan ve kümenin yönetiminde kullanılan controllerEngine ve processorEngine birimleri başlatılmamış olur.

Şekil 3.24'de bir düğümün küme düğüm olarak ayağa kaldırılması örneği verilmiştir.

```
java -cp .:mysql-connector-java-5.1.14-bin.jar tr.edu.yildiz.jstar.test.JStarTest 2000 127.0.0.1 2000
```

Şekil 3. 24 Bir düğümün küme düğüm olarak başlatılması

Küme düğümü başlatılırken kullanılan ilk parametre düğümün kümede hangi kapı üzerinden mesajlaşma yapacağı bilgisini (listenPort) belirtmektedir. İkinci parametre ise mevcut düğüm yöneticisinin IP değerini içerir. Ve sonuncu parametre ise düğüm yöneticisinin kapı değerini gösterir. Burada girilen düğüm yöneticisi ip ve kapı bilgileri kullanılarak ilgili düğüm yöneticisi üzerinden kümeye dahil olunur.

Küme içerisindeki her bir düğüm belirli aralıklarla düğüm yöneticisine aktif olduğunu belirten "ping_client" komutu gönderir. Bu sayede düğüm yöneticisi tarafından kümenin mevcut durumu gözlemlenerek bir problem oluştuğunda gerekli aksiyonların alınması kolaylaşmış ve hızlanmış olur. Aynı zamanda düğüm, düğüm yöneticisinin aktif olduğunu testip etmiş olur. Eğer düğüm yöneticisinin çöktüğü algılanırsa yeni yönetici seçme sürecine geçilir.

Normal şartlar altında kümede bulunan düğümler arasında herhangi bir iletişim olmaz. Düğümler yalnızca düğüm yöneticisi ile iletişim halindedir. Fakat buna rağmen her bir düğüm kümenin tamamı hakkında gerekli bilgiye sahiptir. Normal şartlarda kullanılmayan bu bilgi düğüm yöneticisinin çökmesi durumunda kritik öneme sahip olmaktadır.

Her bir küme düğümün kümedeki konumunu koruyabilmek ve düğüm yöneticisi ile gerekli mesajlaşmayı gerçekleye bilmek için çeşitli birimler kullanmaktadır:

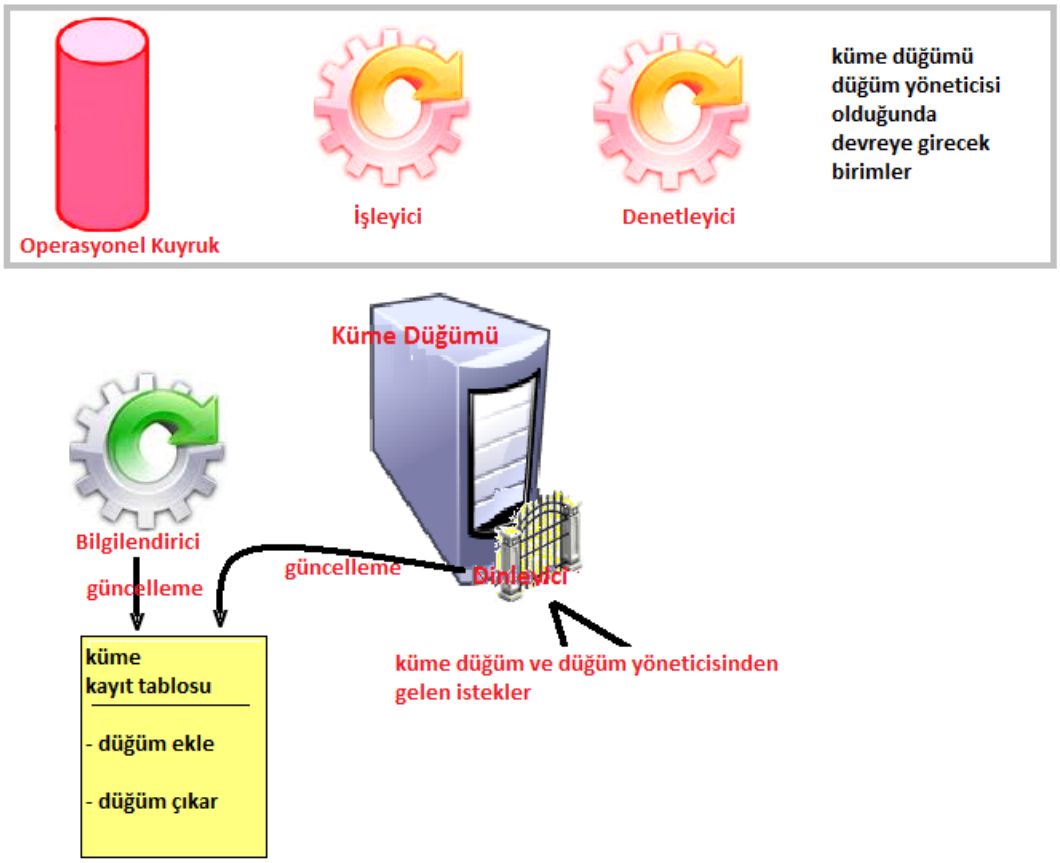
tr.edu.yildiz.jstar.core.network.NodeAcceptor: Küme düğüm yöneticisinden gelen isteklere cevap üreten temel birimdir.. Aynı birim düğüm yöneticisinde de bulunmaktadır. Fakat küme düğümü ile küme düğüm yöneticisine gelen komutlar birbirlerinden farklıdır. Dolayısı ile bir düğüm, düğüm yöneticisi olduğu zaman da aynı birim ile görevine kesintisiz devam edecektir. Tek değişen kendisine gelecek olan komutlar ve bu komutlara üretilecek olan cevaplar olacaktır.

tr.edu.yildiz.jstar.core.network.NodeInformer: Küme düğümünün belirli aralıklarla düğüm yöneticisine “ping_client” komutu göndermesi görevini üstlenen birimidir. Aynı zamanda düğüm yöneticisinin çöktüğü bilgisi bu birim tarafından algılanarak küme düğümünün düğüm yöneticiliğine terfi ettiği ya da yeni bir düğüm yönetisinin seçildiği bilgisi bu birim sayesinde algılanır. Herhangi bir küme düğümünün, düğüm yöneticisi olarak atanması durumunda pasif durumda bulunan “controllerEngine” ve “processorEngine” birimlerini aktifleyerek kendi yaşam döngüsünü sonlandırır.

tr.edu.yildiz.jstar.core.engines.ControllerEngine: Normal şartlarda pasif durumda olup küme düğümü, düğüm yöneticiliğine terfi ettiği durumda “NodeInformer” tarafından aktiflenir.

tr.edu.yildiz.jstar.core.engines.ProcessorEngine: Normal şartlarda pasif durumda olup küme düğüm, düğüm yöneticiliğine terfi ettiği durumda “NodeInformer” tarafından aktiflenir.

Küme düğümü durumundan küme düğüm yöneticiliğine geçiş yapan bir düğüm artık eski görevi olan küme düğümü görevine tekrardan geçemez. Yaşam döngüsü boyunca düğüm yöneticiliği yapmak zorundadır.



Şekil 3. 25 JStar küme düğüm yapısı

3.6.2.1 Düğüm Yöneticisinin Düşmesi

Kümeden gelen komutları işleyerek gerekli aksiyonları alan ve kümeyi denetleyerek oluşan yeni şartlara göre gerekli durumlarda kümeyi tekrardan konfigüre eden düğüm yöneticisinin çökmesi bir çok sistemde oluduğu gibi JStar çatı çözümü için de büyük bir sorun teşkil eder. Düğüm yöneticisinin çökmesinden yeni düğüm yöneticisinin seçilmesine kadar olan süreç içerisinde kümede yapılması gereken bir çok iş bulunmaktadır. Bu işleri şu şekilde özetleyebiliriz:

- Düğüm yöneticisinin çöktüğünün algılanması
- Düğümlerin yeni düğüm yöneticisini seçmesi
- Yeni düğüm yöneticisi seçilene kadar kümenin devamlılığının sağlanması
- Birden fazla düğüm yöneticisinin seçilmesinin engellenmesi
- Bütün düğümler tarafından aynı düğümün düğüm yöneticisi olarak kabul edilmesi

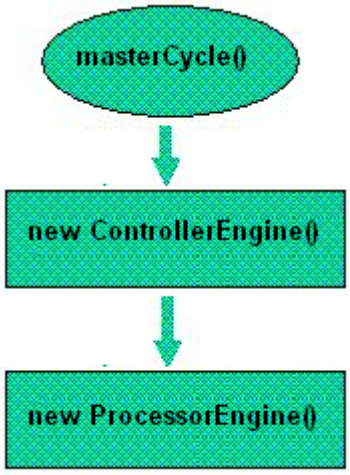
Daha önceden de bahsedildiği üzere kümede bulunan her bir düğüm belirli aralıklarla düğüm yöneticisine bağlantı kurarak ayakta olduğunu belirten “ping_client” komutunu gönderir. Bir düğüm belirtilen zaman aralığında (zamanaşımı) “ping_client” komutunu gönderemediği durumda yani küme düğüm yöneticisi ile bir bağlantı kuramadığında, düğüm yöneticisinin çötüğü sonucuna vararak yeni düğüm yöneticisi seçme sürecine geçer.

Her bir küme düğümü, küme düğüm yöneticisi gibi bütün küme hakkında gerekli bilgilere sahiptir. Kümede hangi id'li küme düğümleri bunduğu, bu düğümler ile hangi IP ve hangi kapı değerleri kullanılarak iletişim kurulabileceği gibi bilgileri “register” (düğümün kümeye giriş yapması) işleminde düğüm yöneticisi tarafından düğüme iletilir. Düğüm yöneticisinin çöktüğünü algılayan düğüm, küme kayıt tablosuna bakarak bir sonraki düğüm yöneticisinin kimin olması gerektiğine karar verir. Buradaki algoritma çok basit olmakla birlikte çok da işlevseldir. Aktif düğümler arasından kümeye ilk dahil olmuş olan düğüm yeni düğüm yöneticisi olarak belirlenir ve bu düğüme “ping_client” komutu gönderilmeye çalışılır.

Hangi düğümün kümeye ilk kayıt olduğu bilgisi nodeID'sinde ifade edilmektedir. NodeID değeri en küçük olan düğüm, kümeye ilk giren düğümdür.

Düğüm yöneticisinin çöktüğünü algılayan bir düğüm yeni düğüm yöneticisi olarak kendisinden farklı bir düğümü tespit ederse kendisini bu yeni düğüm yöneticisine göre ayarlar ve yine zamanaşımı süresi kadar yeni düğüm yöneticisine “ping_client” komutu göndermeye çalışır. Düğüm yöneticisinden “ok.” komutunu aldığı zaman süreç tamamlanmış olur. İlgili küme düğümünden “ok.” komutu alamadığı bir durumda (örneğin düğüm yönetici adayının da çökmüş olması durumu) ilgili düğümün çötüğüne karar verilir ve kayıt tablosundan çıkartılır. Sıradaki en küçük nodeID’li düğüm için aynı süreç tekrarlanarak ve 2*zamanaşımı süresi boyunca yeni düğüm yöneticisine “ping_client” komutu gönderilmeye çalışılır. Bu şekilde yeni düğüm yöneticisi bulunmaya çalışılır. Her bir düğüm yöneticisi adayının zamanaşımı süresi boyunca “ok.” dönmemesi durumunda sıraki düğüm yöneticisi adayına geçilir ve “ping_timeout” süresi zamanaşımı süresi kadar arttılır.

Düğüm yöneticisinin çötüğünü algılayan bir düğüm yeni düğüm yöneticisi olarak kendisini testip etmesi durumunda yani aktif düğümler arasında en küçük nodeID'li düğümün kendisi olması durumunda öncelikli olarak kendi değişkenlerini düğüm yöneticisi olarak günceller. Ve bir düğüm yöneticisinin yapması gereken elzem görevleri yapmak için gerekli olan “controllerEngine” ve “processorEngine”'ları ayağa kaldırarak küme üzerinde gerekli denetsel ve operasyonel işlemlerini gerçekler. Düğümlerden gelen isteklere ilgili cevapları üretir. Şekil 3.26’te ilgili birimlerin nasıl başlatıldığı gösterilmiştir.



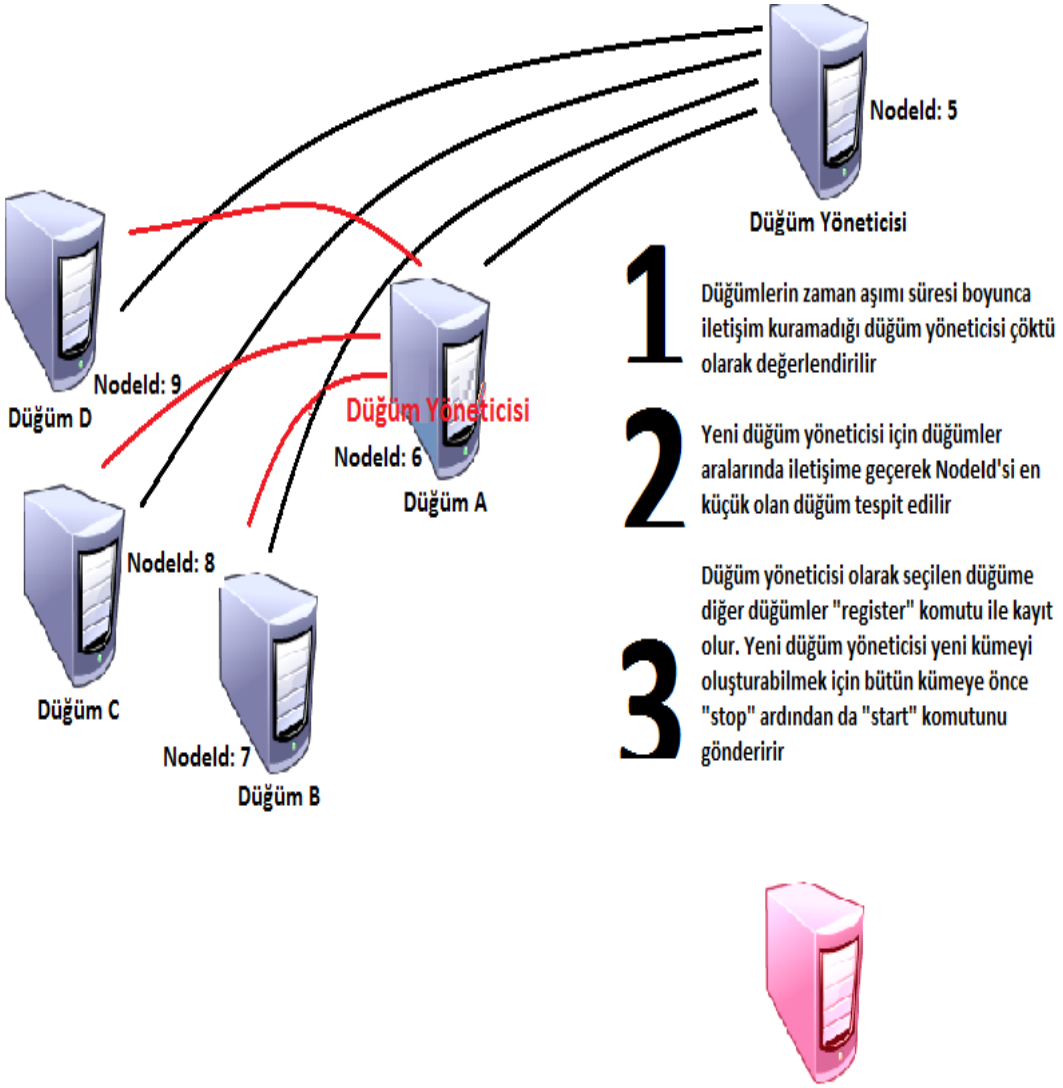
Şekil 3. 26 Bir küme düğümünün düğüm yöneticisi olması

JStar çatı çözümünde her bir düğüm aynı zamanda düğüm yöneticisi görevlerini icra edebilecek şekilde tasarlanarak gerçeklendiğinden dolayı böylesi bir rol geçişi yalnızca bir parametrenin değerine bakılarak kolaylıkla gerçeklenebilecektir. Ve bu şekilde düğüm yöneticisi olarak göreve başlayan bir küme düğümü aynı zamanda düğüm olma görevlerinin hemen hemen tamamını ifa etmeye devam edecektir (artık düğüm yöneticisi olduğu için “ping_server” gibi komutlar göndermek dışında). Burada önem arz eden bir diğer konu da kümeye giriş esnasında düğümlere gerilen nodeID'lerdir. Bu değerlerin tekil olması şarttır. Bu sorunu çözmek için java’nın sunmuş olduğu “senkronizasyon” ve “static” metodolojisinden faydalanılır. Şekil 3.27’de bu çözüm gösterilmiştir. Ayrıca yeni düğüm yöneticisi seçilmesi sürecinde her bir düğüm mevcut nodeID'leri ile devam edeceğinden dolayı herhangi bir tekil sınırlama ihlali problemi oluşmayacaktır.

```
public synchronized long generateNodeId() {
    return ++this.nodeIdCounter;
}
```

Şekil 3. 27 Kümeye dahil olan düğümler için nodeID üretilmesi

Küme düğüm yöneticisinin düşmesi ve yeni düğüm yöneticisinin belirlenmesi süreçleri Şekil 3.28'de özetlenmiştir.



Şekil 3. 28 Küme düğüm yöneticisinin düşmesi ve yeni düğüm yöneticisinin belirlenmesi adımları

3.7 JStar Çatısı Özellikler Dosyası

JStar çatı çözümü genel amaçlı bir çatı olarak tasarlandığı için, bazı değişkenlerin kod dışına çıkarılmasının daha uygun olacağı düşünülmüştür. Bu sayede farklı platformlar için bu parametreler rahatlıkla değiştirilebilecek ve istenen ortama uygun hale

getirilebilecektir. Bu bağlamda ayrıntısı Şekil 3.29'da verildiği gibi olan "jstar.properties" dosyası kullanılmıştır. İlgili dosya jar içerisinde tr.edu.yildiz.jstar.core.conf paketi altında bulunmaktadır.

```
TIME_START_PING  = 10000
TIME_EXPIRE_PING = 30000
TIME_EXPIRE_MANAGER_RESPONSE = 30000
DEFAULT_CONN_TIMEOUT  = 60000
PERIOD_PING = 7000
PROCESSOR_SLEEP_DURATION = 10000
CONTROLLER_SLEEP_DURATION = 5000
HTTP_CONNECTION_TIMEOUT = 20000
HTTP_READ_TIMEOUT = 30000
HTTP_RETRY_COUNT = 3
HTTP_TOTAL_CONN_COUNT = 20
HTTP_CLINT_MAX_CONN_COUNT = 10
```

Şekil 3. 29 JStar özellikler dosyası değişkenleri

Burada geçen parametrelerin açıklamaları şu şekildedir:

TIME_START_PING: Milisaniye cinsinden olup küme düğüm yöneticisi tarafından küme düğümüne tanımlanmış olan iletişim zamanaşımı süresini belirtir. Herhangi bir küme düğümü TIME_START_PING süresi boyunca küme düğüm yöneticisiyle iletişim kuramıyor ise düğüm yöneticisi küme düğümü ile iletişime geçer. Varsayılan süre 10 saniye olarak tanımlanmıştır.

TIME_EXPIRE_PING: Milisaniye cinsinden olup küme düğüm yöneticisi ile bir küme düğümü arasındaki maksimum mesajlaşamama süresini belirtir. TIME_START_PING süresi boyunca küme düğümünden haber alamayan küme düğüm yöneticisi TIME_EXPIRE_PING süresi boyunca küme düğümü ile iletişim kurmaya çalışır. Bu sürede boyunca hala küme düğümünden cevap alınamıyor ise ilgili düğümün çötüğü düşünülerek kümeden çıkartılması için gerekli içsel komut üretilir. Varsayılan süre 30 saniye olarak tanımlanmıştır.

TIME_EXPIRE_MANAGER_RESPONSE: Milisaniye cinsinden olup bir küme düğümünün küme yöneticisine tanımış olduğu zamanaşımı süresini belirtir. Bu süre boyunca küme yöneticisi ile iletişim kuramayn küme düğümü, yöneticinin çökmüş olduğuna karar vererek yeni küme yönetici seçimi sürecine girer. Varsayılan süre 30 saniye olarak tanımlanmıştır.

DEFAULT_CONN_TIMEOUT: Milisaniye cinsinden olup priz bağlantı zamanaşımı süresini belirtir. Varsayılan süre 60 saniye olarak tanımlanmıştır.

PERIOD_PING: Milisaniye cinsinden olup küme düğümünün bir döngüyü tamamladıktan sonra bekleyeceği süreyi belirtir. Yani iş parçacığı uyku süresidir. Varsayılan süre 7 saniye olarak tanımlanmıştır.

PROCESSOR_SLEEP_DURATION: Milisaniye cinsinden olup küme düğüm yöneticisinde bulunan ProcessorEngine'ın bir döngüyü tamamladıktan sonra bekleyeceği süreyi belirtir. Yani iş parçacığı uyku süresidir. Varsayılan süre 10 saniye olarak tanımlanmıştır.

CONTROLLER_SLEEP_DURATION: Milisaniye cinsinden olup küme düğüm yöneticisinde bulunan ControllerEngine'ın bir döngüyü tamamladıktan sonra bekleyeceği süreyi belirtir. Yani iş parçacığı uyku süresidir. Varsayılan süre 5 saniye olarak tanımlanmıştır.

HTTP_CONNECTION_TIMEOUT: Milisaniye cinsinden olup HTTP adaptörü tarafından kullanılan bir parametredir. İlgili HTTP sunucuya bağlantı kurarken belirtilen zamanaşımı süresidir. Yani bağlantı kurabilmek için maksimum bu süre kadar beklenecektir. Varsayılan süre 20 saniye olarak tanımlanmıştır.

HTTP_READ_TIMEOUT: Milisaniye cinsinden olup HTTP adaptörü tarafından kullanılan bir parametredir. İlgili HTTP sunucusuna bağlantı kurulduktan sonra veri okumak için belirtilen zamanaşımı süresidir. Varsayılan süre 30 saniye olarak tanımlanmıştır.

HTTP_RETRY_COUNT: HTTP adaptörü tarafından kullanılan bir parametredir. İlgili HTTP sunucusuna bağlantı kurmak için maksimum kaç kez tekrar deneme edileceğini belirtir. Varsayılan değer 3 olarak tanımlanmıştır.

HTTP_TOTAL_CONN_COUNT: HTTP adaptörü tarafından kullanılan bir parametredir. İlgili HTTP bağlantı havuzunda en fazla kaç bağlantı bulunabileceğini gösteren parametredir. Varsayılan değer 20 olarak tanımlanmıştır.

HTTP_CLINT_MAX_CONN_COUNT: HTTP adaptörü tarafından kullanılan bir parametredir. İlgili HTTP bağlantı havuzunda makina başına en fazla kaç bağlantı bulunabileceğini gösteren parametredir. Varsayılan değer 10 olarak tanımlanmıştır.

BÖLÜM 4

JSTAR JAVA ÇATI ADAPTÖRLERİ

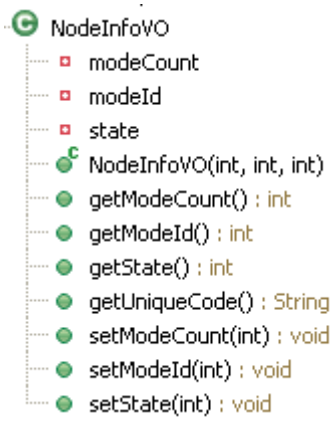
JStar çatısı temel olarak uygulama geliştiriciye 3 farklı bilgi sunmaktadır. Bu bilgiler ve açıklamaları şu şekildedir;

modeCount: Kümede kaç adet aktif düğüm bulunduğu bilgisini içerir ve veriye erişimde kullanılır.

modeId: Veriye erişimde hangi kayıtların alınacağını belirten bir ID olarak kullanılır ve 0 ile modeCount-1 arasında herhangi bir değeri alabilir.

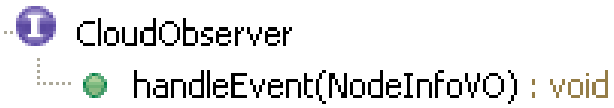
state: Küme düğümünün hangi durumda olduğunu belirten bir değerdir ve STATE_INIT, STATE_WAIT ve STATE_GO değerlerinden yalnızca birini alabilir. Veri erişimi için durumun STATE_GO olması gereklidir. Aksi koşulda durum değişene kadar veri erişimi durdurulur.

Uygulama geliştiricisine bu 3 veri tipi tr.edu.yildiz.jstar.core.models.NodeInfoVO modeli ile iletilmektedir. Şekil 4.1'de NodeInfoVO modelinin yapısı verilmiştir.



Şekil 4. 1 Uygulama geliştiricisine iletilen ve düğüm hakkında gerekli bilgileri içeren veri modeli

Herhangi bir Java sınıfı içerisinde düğüm bilgisinin alınabilmesi için sınıfın Şekil 4.2'de ayrıntısı verilen tr.edu.yildiz.jstar.shell.intf.CloudObserver arabiriminin handleEvent metodunu uygulaması gerekir.



Şekil 4.2 Düğüm bilgisini iletmekte kullanılan arabirimin yapısı

Ayrıca dinleyici sınıftın tekil dizayn deseni ile geliştirilmiş olan tr.edu.yildiz.jstar.shell.facade.CloudNotifier objesine kayıt edilmesi gerekmektedir.

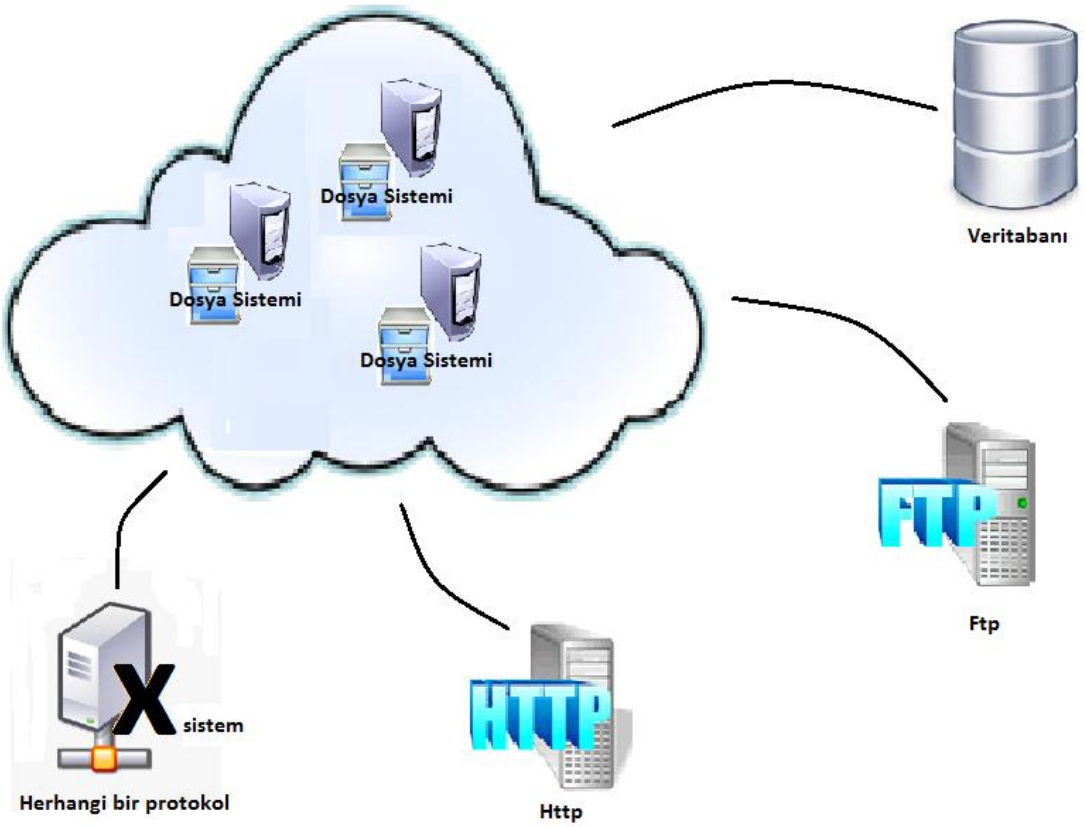
Bu şekilde kullanılacak bir küme dinleyicisi küme hakkında gerekli duyulan "modeCount", "modeID" ve "state" bilgilerini NodeInfoVo objesi ile alacaktır. Ve kümede oluşan her bir değişiklik (state, modeCount ya da modeID) handleEvent metodunu tetikleyerek ilgili objeyi güncelleyecektir. Böylelikle geliştiriciye hangi durumlarda veriye nasıl ulaşması gerektiği ile ilgili bütün bilgiler sunulmuş olacaktır.

En temel hali ile kullanım şekli yukarıda özetlendiği gibi olan JStar çatı çözümü bir çok farklı amaç için kullanılabilir. Temelde kaynak bağımlı uygulamaların küme düzleminde çalıştırılması için tasarlanmış olan JStar çatı çözümü aslında çok daha geniş bir kullanım alanı sunmaktadır.

Buradaki kullanım genişliği uygulama geliştiricilerin hayal gücü ile sınırlı olmakla birlikte bizim çalışmamızın ana noktasını oluşturan temel kullanım alanlarını aşağıdaki gibi özetlemek mümkündür:

- veritabanı kaynaklarına erişim
- dosya sistemi kaynaklarınıa erişim
- ftp kaynaklarına erişim
- http kaynaklarına erişim

Bu kaynak sistemler JStar çatı çözümünün temel hedef alanı olduğundan dolayı, yukarıdaki gibi bir temel erişim yerine geliştirlen adaptörler üzerinden bu işlemlerin daha basit ve sağlıklı bir hala dönüştürülmesi için 4 farklı kaynak tipi (veritabanı, dosya sistemi, ftp, http) için yine 4 farklı adaptör geliştirilmiştir ve uygulama geliştiricilere sunulmuştur. Şekil 4.3'te bu durum özetlenmiştir.



Şekil 4.3 JStar kümesi ve kaynak sistemler

4.1 Veritabanı Adaptörü

Bir çok farklı veritabanı sürümü olmasından dolayı (mysql, oracle, db2, microsoft sql server, vb...) bu çalışma kapsamında yalnızca mysql veritabanı için adaptör

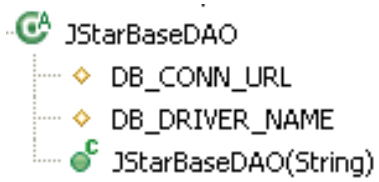
geliştirilmiştir. Diğer veritabanı çeşitleri için de benzer adaptörler çok rahatlıkla geliştirilerek açık kaynak olarak sunulan JStar çatı çözümüne eklenebilir.

Daha önceden de belirtildiği gibi JStar çatı çözümünün amacı verinin alınması ile ilgili sıkınıtıları çözmek olarak özetlenebilir. Verinin silinmesi, güncellenmesi gibi durumlar zaten verinin okunmasından sonra gerçekleşebilecek işlemler olduğundan dolayı asıl sorun küme düzeninde hangi düğümün hangi veri ile ilgilenmesi gerektiğinin tespit edilmesidir. Bu nedenle veritabanı adaptörü verinin okunmasına (yani "select" sorgusuna) dönük çözüm sunmaktadır.

JStar çatı çözümü kapsamında MySql veritabanı sunucusuna erişimde yaygın bir kullanım ağı bulunan mysql-connector-java-X.jar kütüphanesi kullanılmıştır.

4.1.1 Veritabanı Adaptör Yapısı

JStar MySql veritabanı adaptörü JStarBaseDAO sınıfından türetilmektedir. Bu sınıf içerisinde MySql veritabanı ile ilgili temel bağlantı bilgileri tutulur. Sınıfın yapısı Şekil 4.4'te verildiği gibidir.

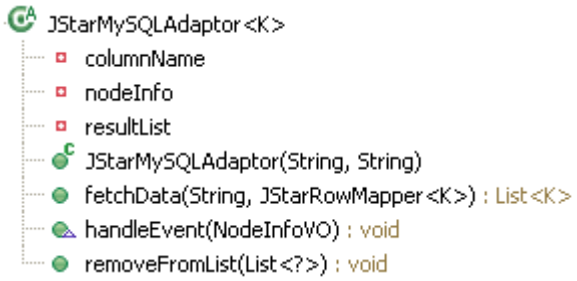


Şekil 4.4 MySql bağlantı bilgilerinin tutuluduğu sınıfı

JStarBaseDAO veri erişim objesini genişleterek geliştirilen soyut JStarMySQLAdaptor sınıfı uygulama geliştiricilere açılan asıl sınıftır. Aynı zamanda kullanılan jenerik tanımlaması ile maliyetli olan tip dönüştürme işleminden kurtulmuş olunur. JStarMySQLAdaptor sınıfının yapısı Şekil 4.6'da verildiği gibidir.

JStarMySQLAdaptor aynı zamanda CloudObserver objesini uygulayarak kümenin durumunu, veri erişim için gerekli olan NodeInfoVO model objesini ve kümedeki değişimleri takip etmekte ve küme ile geliştirici arasında bir katman oluşturmaktadır. Kümedeki her bir değişim handleEvent metodu aracılığı ile adaptöre düşecektir.

Sınıfın oluşturucu kısmında veri erişim kriterlerinin uygulanacağı tablo kolon ismi paramatre olarak istenmektedir.



Şekil 4.5 JStarMySQLAdaptor adaptörünün yapısı

Uygulama geliştiricilere sunulan "fetchData" metoduna bakacak olursak, iki farklı parametre aldığını ve sonuçta vermiş olduğumuz jenerik tipinde bir liste objesi döndüğünü görürüz. Birinci parametre uygulama geliştiricisinin veri almak için yazacağı bir sorgu cümleciği olacaktır. Bu sorgu cümleciği adaptör tarafından çeşitli modifikasyonlara uğratılacak ve yeniden dizayn edilecektir. Böylelikle ilgili tablonun bütün kayıtları değil de yalnızca ilgili düğümün alması gerekli olan kayıtlar getirilecektir.

Örneğin Şekil 4.6'da verilen sorgu cümleciği adaptör aracılığı ile Şekil 4.7'de gösterildiği gibi güncellenmiştir. (modeCount'un 5, modeID'nin 3 olduğu bir ortamda bu çıktı elde edilmiştir)

```
select * from my_table
```

Şekil 4.6 Örnek bir sorgu cümleciği

```
select * from my_table where id % 5 = 3
```

Şekil 4.7 Örnek bir sorgu cümleciğinin adaptör aracılığı ile değiştirilmesi

Bu sayede bütün kayıtlar yerine yalnızca küme düğümünün ihtiyaç duyduğu kayıtlar alınmış olunacaktır. Diğer kayıtlar kümede bulunan diğer düğümler tarafından paylaşılacaktır. Böylelikle modern geliştirme yöntemlerinden olan yük dengeleme gerçeklenmiş olacaktır.

Bu yapı tamamen dinamik olduğundan dolayı kümeye yeni bir düğümün eklenmesine ya da kümede bulunan mevcut düğümlerden birinin düşmesine anında tepki verilebilecek ve sistemin sürekli ve güvenli bir şekilde çalışması sağlanmış olacaktır.

Bu sorgu cümleciğinin çalışması sonucunda dönen sonuç kümesinin ilgili objeye aktarılması için ikinci parametrede JStarRowMapper<K> şeklinde bir adresleyici

istenmektedir. Burada kullanılacak olan adresleyici JstarRowMapper<K> arabirimini uygulayarak gerçeklenmek zorundadır. Böylelikle maliyetli bir iş olan tip dönüşümü işleminden kurtulmuş olunmaktadır. JStarRowMapper arabiriminin yapısı Şekil 4.8'de verilmiştir.

Şekil 4.8 JStarRowMapper arabirim yapısı

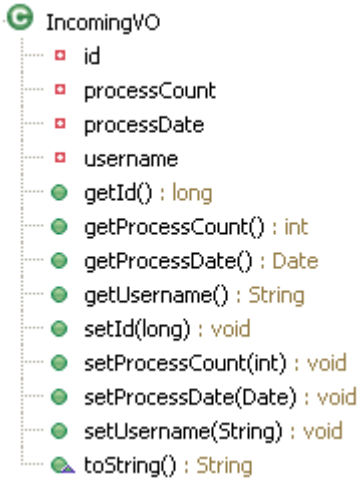
4.1.2 Veritabanı Adaptörünün Kullanım Şekli

Örneğimizde kullanılacak olan veritabanı tablosunun yapısı Şekil 4.9'da verilmiştir.

```
CREATE TABLE `test_incoming1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(100) DEFAULT NULL,  select * from my_table
  `process_date` datetime DEFAULT NULL,
  `process_count` int(11) DEFAULT '0',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=6083 DEFAULT CHARSET=latin1;
```

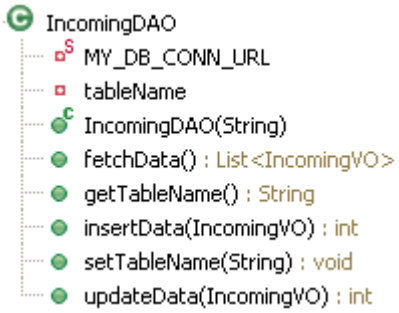
Şekil 4.9 "test_incoming1" tablosunun yapısı

Şekil 4.10'da verilen veritabanı tablosunun java model karşılığı Şekil 4.10'da gösterildiği gibidir.



Şekil 4.10 IncomingVO model yapısı

Veri tabanından veri almak için kullanılan veri erişim objesi (DAO) yapısı Şekil 4.11'de gösterilmiştir.



Şekil 4.11 IncomingVO için DAO

Dönecek olan veri tipinin belirlenmesi için JStarRowMapper uygulanmış olup ayrıntısı Şekil 4.12'de verilmiştir.

Şekil 4.12 Dönecek olan verinin atanmasında kullanılan adresleyici

Burada dikkat edilmesi gereken en önemli noktalardan birisi de veri işleme işlemi tamamlandıktan sonra DAO objesinin removeFromList metodunun çağrılması gerekliliğidir. Veritabanından çekilen veri işlenmeden önce kümede bir durum ve dolayısı ile bir kip değişim süreci yaşanması durumunda elimizde bulunan verinin bize ait olmadığı durumu ortaya çıkmaktadır. Bu sorunu gidermek için mevcut veri bir referans ile adaptörde de tutulmaktadır ve kümede yaşanacak durum değişiminde mevcut referanlı veriler silinecektir. Böylelikle bize ait olmayan verinin bizim tarafımızdan işlenmesi önlenmiş olacaktır.

Bu durumda her bir listenin bir de adaptörde karşılığı olacağından dolayı liste ile işimizin bitmesi durumunda adaptörden bu referansı kaldırmamız gerekecektir. Bu işlem için removeFromList metodunu çağırmamız gerekmektedir. Aksi durumda veri referans listesinden kaldırılmadığından dolayı Java çöp toplayıcısı tarafından da silinemeyerek bellek kaçağı oluşması sorunu doğacaktır.

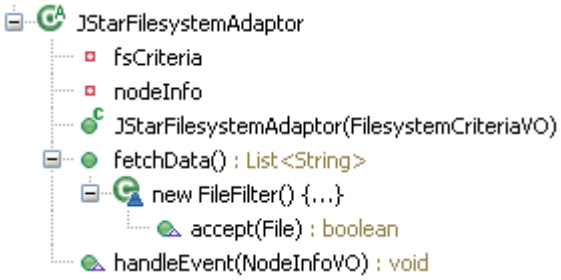
4.2 Dosya Sistemi Adaptörü

Dosya sistemi adaptörü aracılığı ile bir dizin altındaki dosyalar küme kullanımına açılmış olur ve böylelikle birden fazla düğümün paralel olarak kaynaklara erişimi sağlanmış

olur. Bu şekilde bir kaynak erişimi bize performans, güvenilirlik ve erişilebilirlik kazandıracaktır.

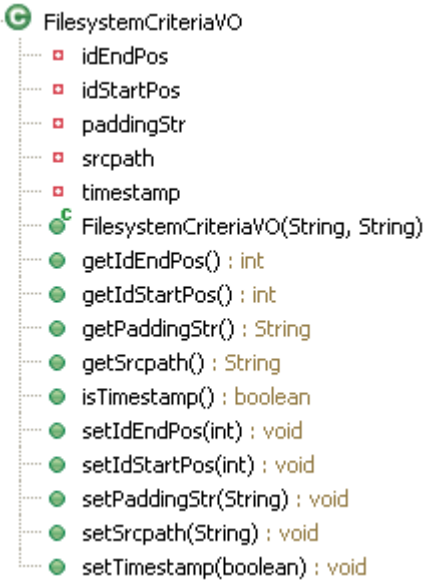
4.2.1 Dosya Sistemi Adaptör Yapısı

JStar çatı çözümü dosya sistemi tabanlı kaynak erişiminde de uygulama geliştiricilere hazır çözüm sunmaktadır. tr.edu.yildiz.jstar.shell.adaptor.fs paketi altında bulunan soyut JStarFilesystemAdaptor sınıfı kümeden alınan mesajları dinleyecek ve uygulayacak şekilde tasarlanmıştır. Şekil 4.13'te JStarFilesystemAdaptor sınıfının yapısı verilmiştir.



Şekil 4. 13 JStarFilesystemAdaptor sınıfının yapısı

Adaptörün oluşturucu parametresi olarak kullanılan FilesystemCriteriaVO objesi dosya sitemi ve dosya sisteminde bulunan veri hakkında gerekli bilgileri içermektedir. İlgili sınıfın yapısı Şekil 4.14'te verilmiştir.



Şekil 4.14 FilesystemCriteriaVO modelinin yapısı

FilesystemCriteriaVO ile aşağıdaki kriterler belirtilebilir:

timestamp: Kaynak sistemin kümede nasıl paylaşılacağı bilgisini belirtir. Varsayılan değeri true olup dosyanın yaratılma zaman bilgisinden yararlanılacağı anlamına gelmektedir.

Srcpath: Kaynağın dosya sistemi üzerinde hangi dizinden alınacağı bilgisinini içerir. Örnek bir srcpath: "/Users/ysnky/test/jstar_test/" şeklinde olabilir.

idStartPos: Kaynak erişimi için dosya oluşturulma zaman bilgisi yerine dosya içerisinde geçen bir “id” değerinden faydalanılmak istenirse “timestamp” parametresi false yapılır ve idStartPos değeri dosya isminde geçen id'nin başlangıç pozisyonunu belirtir.

idEndPos: Kaynak erişimi için dosya oluşturulma zaman bilgisi yerine dosya içerisinde geçen bir “id” değerinden faydalanılmak istenirse “timestamp” parametresi false yapılır ve idEndPos değeri dosya isminde geçen id'nin bitiş pozisyonunu belirtir.

paddingStr: Erişilen kaynağın işlendiğini belirtmek için dosya ismi güncellenir. PaddingStr ile belirtilen değer güncelleme işleminde kullanılır. Örnek bir paddingStr: "pro_" şeklinde olabilir ve bu durumda işlenen dosyaların isimlerinin başına “pro_” ön takısı eklenir.

4.2.2 Dosya Sistemi Adaptörünün Kullanım Şekli

Geliştirilecek olan dosya sistemi DAO sınıfı JStarFilesystemAdaptor sınıfından türetilmek zorundadır. Yazılacak örnek bir uygulamada ise FilesystemCriteriaVO model objesi ilgili dosya sistemi bilgileri geçildikten sonra DAO objesinin oluşturucusuna parametre olarak geçilmelidir. DAO objesinin fetchData metodu kullanılarak VO aracılığı ile belirtilen dosya sistemi kaynak verilerine ulaşılır.

4.3 FTP Adaptörü

FTP adaptörü dosya sistemi adaptörü ile benzerlik göstermektedir. Dosya sistemi adaptöründe olduğu gibi FTP adaptöründe de bir kaynak üzerinde bulunan dosyalara erişilir. Düğümlerin bu erişim esnasında yalnızca kendileri ile ilgili olan dosyaları okumaları ve işlemeleri gerekmektedir. Yetkileri olmayan dosyalara herhangi bir erişim çok büyük bir problem olan verinin birden fazla işleme girmesi gibi tehlikeli sonuçlar doğuracaktır.

JStar çatı çözümü kapsamında FTP sunuculara erişimde yaygın bir kullanım ağı bulunan apache commons-net.jar kütüphanesi kullanılmıştır.

4.3.1 FTP Adaptör Yapısı

Dosya sistemi adaptörü ile FTP adaptörü arasındaki temel fark, dosya sisteminde kaynak olarak belirtilen dizin düğümler ile aynı makina üzerinde bununmakta iken FTP adaptöründe kaynak farklı bir makina üzerinde bulunur ve her bir düğüm ilgili kaynağa erişebilmek için FTP protokolünü kullanmak zorundadır.

FTP ve kaynak erişimi için gerekli kriterler Şekil 4.15'te yapısı verilen FTPCriteriaVO ile belirtilir.

Şekil 4.15 FTPCriteriaVO modelinin yapısı

FTPCriteriaVO modelinde geçen alanlar ve açıklamaları şu şekildedir:

timestamp: Kaynak sistemin kümede nasıl paylaşılacağı bilgisini belirtir. Varsayılan değeri true olup dosyanın yaratılma zaman bilgisinden yararlanılacağı anlamına gelmektedir.

Srcpath: Kaynağın dosya sistemi üzerinde hangi dizinden alınacağı bilgisinini içerir. Örnek bir srcpath: "/Users/ysnky/test/jstar_test/"

idStartPos: Kaynak erişimi için dosya yaratılma zamanı değeri yerine dosya içerisinde geçen bir id değerinden faydalanılmak istenirse “timestamp” parametresi false yapılmalı ve idStartPos değerine dosya isminde geçen id'nin başlangıç pozisyonunu girilmelidir.

idEndPos: Kaynak erişimi için dosya yaratılma zamanı değeri yerine dosya içerisinde geçen bir id değerinden faydalanılmak istenirse “timestamp” parametresi false yapılmalı ve idEndPos değerine dosya isminde geçen id'nin bitiş pozisyonunu girilmelidir.

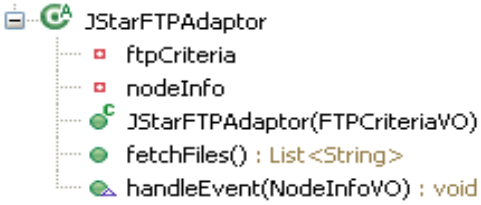
paddingStr: Erişilen kaynağın işlendiğini belirtmek için dosya ismi güncellenir. PaddingStr’de belirtilen değer yeni dosya isminde kullanılır.

server: Ftp yapılacak sunucunu IP ya da DNS ismi belirtir.

username: Ftp protokolünde kullanılan erişim için gerekli kullanıcı ismi bilgisini belirtir.

password: Ftp protokolünde kullanılan ve erişim için gerekli şifre bilgisini belirtir.

Şekil 4.16’da uygulama geliştiricilere sunulan soyut JStarFTPAdaptor sınıfının yapısı verilmiştir. Dikkatle incelendiğinde dosya sistemi adaptörü ile olan benzerliklerinin dikkat çekici olduğu gözlemlenecektir. Dosya sistemi adaptöründe bahsetmiş olduğumuz kural ve kriterlerin büyük kısmı FTP adatörü için de geçerlidir.



Şekil 4.16 FTP adaptör yapısı

4.3.2 FTP Adaptörünün Kullanım Şekli

Geliştirilecek olan DAO sınıfı JStarFTPAdaptor sınıfından türetilmek zorundadır. Ayrıca DAO sınıfı oluşturucusuna FTPCriteriaVO tipinde ve gerekli FTP bilgilerini içeren obje parametre olarak verilmelidir. DAO objesinin fetchFiles metodu kullanılarak VO aracılığı ile belirtilen FTP kaynak sistem verilerine ulaşılır.

4.4 HTTP Adaptörü

HTTP protokolü üzerinden kaynak erişimine ihtiyaç duyulan sistemler düşünülerek geliştirilmiş. JStar çatı çözümü kapsamında http istekleri için yaygın bir kullanım ağı olan apache http client kütüphanesi kullanılmıştır. Bu kapsamda http adaptörü kullanacak uygulamaların sınıfyolunda aşağıdaki jar'ların bulunması gerekmektedir.

- commons-codec-X.jar
- commons-httpclient-X.jar
- commons-logging-X.jar

4.4.1 HTTP Adaptör Yapısı

HTTP çağrıları ve kaynak erişimi için gerekli kriterler Şekil 4.17'de yapısı verilen HTTPCriteriaVO ile belirtilir.

Şekil 4. 17 HTTPCriteriaVO modelinin yapısı

HTTPCriteriaVO modelinde geçen alanlar ve açıklamaları şu şekildedir:

url: Http çağrısı yapılacak adresi belirtir.

method: “http post” mu yoksa “http get” mi yapılacağı bilgisini yani http çağrısının metot bilgisini gösterir.

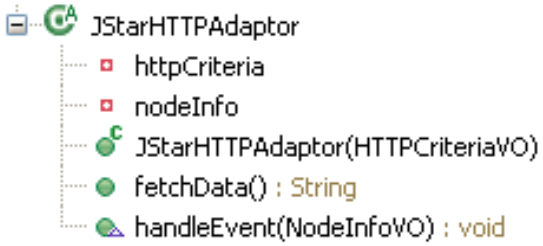
postBody: Http post’da gövdeye yazılacak olan veriyi gösterir.

params: Yapılacak olan http çağrısında kullanılacak parametre listesini belirtir.

proxyHost: Http bir vekil arkasından yapılıyor ise ilgili vekil sunucusunun IP bilgisi girilmelidir.

proxyPort: Http bir vekil arkasından yapılıyor ise ilgili vekil sunucusunun kapı bilgisi girilmelidir.

Şekil 4.18’de uygulama geliştiricilere sunulan soyut JStarHTTPAdaptor sınıfın yapısı verilmiştir.



Şekil 4. 18 HTTP adaptör yapısı

4.4.2 HTTP Adaptörünün Kullanım Şekli

Geliştirilecek olan DAO sınıfı JStarHTTPAdaptor sınıfından türetilmek zorundadır. Ayrıca DAO sınıfının oluşturucusuna HTTPCriteriaVO tipinde ve gerekli HTTP bilgilerini içeren obje parametre olarak verilmelidir. DAO objesinin fetchData metodu kullanılarak VO aracılığı ile belirtilen HTTP kaynak sistem verilerine ulaşılır.

BÖLÜM 5

JSTAR KÜME VE ADAPTÖR TESTLERİ

Bu bölümde JStar çatı çözümü kullanılarak kümenin kurulumu ve adaptörler ile veri erişim testleri gerçekleştirilecektir. Veri erişimde veritabanı olarak MySql sunucusu kullanılacağı için yalnızca mysql bağlantı jar'ı sınıf yoluna eklenmiştir. Geliştirilen örnek uygulamanın sınıf yolu şu şekildedir:

- jstar-1.0.jar
- mysql-connector-java-5.1.14-bin.jar

Uygulama testleri Mac OS X platformunda örnek bir uygulama yazılarak gerçekleştirilmiştir. Veri tabanı olarak MySql sunucusu, veritabanı istemcisi olarak ise Sequel Pro kullanılmıştır.

5.1 JStar Küme Testleri

Şekil 5.1'de verildiği gibi uygulama küme düğüm yöneticisi kipinde başlatılarak yeni bir küme oluşturulmuştur.

```
# java -cp .:mysql-connector-java-5.1.14-bin.jar:jstar-1.0.jar:jstar_test.jar com.ysnky.jstartest.JStarTest 2000
.................ObserverNotifier.....................
=====================Jstar Server v1.0=====================
Server starts to listening at port: 2000
.................SystemVar start up.....................
new message
observer size:0
============id:1, type:master
============id:1, type:master
============id:1, type:master
============id:1, type:master
```

Şekil 5. 1 JStar kümesinin oluşturulması

Şekil 5.2'de verildiği gibi küme düğüm yöneticisine uç birim aracılığı ile bağlanılarak kümenin durumu hakkında bilgi alınabilir. "get_list" komutu kullanılarak kümede bulunan bütün düğümler hakkında bilgi alınabilmektedir.

```
# telnet 127.0.0.1 2000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[[get_list]]

----------------------------------------NODE----------------------------------------
id:1
ip:127.0.0.1
port:2000
type:master
mode:0/1
lastupdate:1307830555785
------------------------------------------------------------------------------------
```

Şekil 5. 2 Kümenin yapısı hakkında bilgi almak için uç birim kullanılır

Şekil 5.3'te kümeye yeni bir düğümün nasıl eklendiği bilgisi verilmektedir. Burada alınan hatanın nedeni kayıt işleminin gerçekleştirilemeden verinin alınması işlemine başlanmaya çaşılmasıdır. Bu durum herhangi bir sıkıntı oluşturmayacaktır ve kayıt işleminden sonra veri alımı gerçekleştirilebilecektir.

```
# java -cp .:mysql-connector-java-5.1.14-bin.jar:jstar-1.0.jar:jstar_test.jar com.ysnky.jstartest.JStarTest 2001 127.0.0.1 2000
.................ObserverNotifier.....................
====================Jstar Server v1.0====================
Server starts to listening at port: 2001
.................SystemVar start up.....................
sending... ip:127.0.0.1, port: 2000, cmd: [[register][2001]]
Message: [[ok.][3]]
Connection closed...
java.lang.NullPointerException
        at tr.edu.yildiz.jstar.shell.adaptor.db.JStarMySQLAdaptor.fetchData(JStarMySQLAdaptor.java:44)
        at tr.edu.yildiz.jstar.test.IncomingDAO.fetchData(IncomingDAO.java:28)
        at tr.edu.yildiz.jstar.test.DBProcessor.processData(DBProcessor.java:30)
        at tr.edu.yildiz.jstar.test.DBProcessor.run(DBProcessor.java:21)
New Client Accepted: 127.0.0.1
inputLine: [[stop]]
cmd: stop
Connection closed...
New Client Accepted: 127.0.0.1
inputLine: [[start][2][1][1,127.0.0.1,2000,0,1307830555785][3,127.0.0.1,2001,1,1307830724753]]
cmd: start
new message
observer size:1
Connection closed...
sending... ip:127.0.0.1, port: 2000, cmd: [[ping_client][3]]
Message: [[ok.]]
Connection closed...
```

Şekil 5. 3 Kümeye yeni bir düğüm eklenmesi

Şekil 5.4'te verildiği gibi küme düğüm yöneticisine uç birim aracılığı ile bağlanılarak kümenin durumu hakkında bilgi alınabilir.

```
# telnet 127.0.0.1 2000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[[get_list]]

---------------------------------------NODE----------------------------------------
id:1
ip:127.0.0.1
port:2000
type:master
mode:0/2
lastupdate:1307830555785
-------------------------------------------------------------------------------


---------------------------------------NODE----------------------------------------
id:3
ip:127.0.0.1
port:2001
type:slave
mode:1/2
lastupdate:1307830745760
-------------------------------------------------------------------------------
```

Şekil 5. 4 Uç birim kullanılarak küme listesi alınıyor

Şekil 5.5'te küme yöneticisi ve düğümler arasında gerçekleşen mesajlaşmalar gösterilmiştir.

```
observer size:1
sending... ip:127.0.0.1, port: 2001, cmd: [[stop]]
Message: [[ok.]]
Connection closed...
new message
observer size:1
formated cmd:[[start][2][1][1,127.0.0.1,2000,0,1307830555785][3,127.0.0.1,2001,1,1307830724753]]
sending... ip:127.0.0.1, port: 2001, cmd: [[start][2][1][1,127.0.0.1,2000,0,1307830555785][3,127.0.0.1,2001,1,1307830724753]]
Message: [[ok.]]
Connection closed...
new message
observer size:1
===========id:1, type:master
===========id:3, type:slave
id:3, type:1,time diff:1065
===========id:1, type:master
===========id:3, type:slave
id:3, type:1,time diff:6066
New Client Accepted: 127.0.0.1
inputLine: [[ping_client][3]]
```

Şekil 5. 5 Küme mesajlaşma örnekleri

Şekil 5.6'da kümeye ikinci düğüm ekleniyor. Bu düğüm ile birlikte kümede bir düğüm yöneticisi (ki her zaman bulunur ve yalnızca bir tane olur) ve iki adet düğüm bulunmaktadır.

```
# java -cp .:mysql-connector-java-5.1.14-bin.jar:jstar-1.0.jar:jstar_test.jar com.ysnky.jstartest.JStarTest 2002 127.0.0.1 2000
.................ObserverNotifier.....................
======================Jstar Server v1.0======================
Server starts to listening at port: 2002
.................SystemVar start up.....................
sending... ip:127.0.0.1, port: 2000, cmd: [[register][2002]]
Message: [[ok.][4]]
Connection closed...
java.lang.NullPointerException
        at tr.edu.yildiz.jstar.shell.adaptor.db.JStarMySQLAdaptor.fetchData(JStarMySQLAdaptor.java:44)
        at tr.edu.yildiz.jstar.test.IncomingDAO.fetchData(IncomingDAO.java:28)
        at tr.edu.yildiz.jstar.test.DBProcessor.processData(DBProcessor.java:30)
        at tr.edu.yildiz.jstar.test.DBProcessor.run(DBProcessor.java:21)
New Client Accepted: 127.0.0.1
inputLine: [[stop]]
cmd: stop
Connection closed...
New Client Accepted: 127.0.0.1
inputLine: [[start][3][2][1,127.0.0.1,2000,0,1307830555785][3,127.0.0.1,2001,1,1307830780776][4,127.0.0.1,2002,1,1307830784720]]
cmd: start
new message
observer size:1
Connection closed...
sending... ip:127.0.0.1, port: 2000, cmd: [[ping_client][4]]
Message: [[ok.]]
Connection closed...
```

Şekil 5. 6 Kümeye 2.düğümün eklenmesi

Kümeye 2.düğüm eklendikten sonra alınan küme listesi Şekil 5.7'de verilmiştir.

```
# telnet 127.0.0.1 2000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[[get_list]]

----------------------------------------NODE----------------------------------------
id:1
ip:127.0.0.1
port:2000
type:master
mode:0/3
lastupdate:1307830555785
--------------------------------------------------------------------------------


--------------------------------------NODE--------------------------------------
id:3
ip:127.0.0.1
port:2001
type:slave
mode:1/3
lastupdate:1307830801781
--------------------------------------------------------------------------------


--------------------------------------NODE--------------------------------------
id:4
ip:127.0.0.1
port:2002
type:slave
mode:2/3
lastupdate:1307830805727
--------------------------------------------------------------------------------
```

Şekil 5. 7 Kümeye 2. düğümün eklenmesinden sonra alınan küme listesi

Şekil 5.8'de kümeye üçüncü düğüm ekleniyor. Bu düğüm ile birlikte kümede bir düğüm yöneticisi ve üç adet düğüm bulunmaktadır.

```
# java -cp .:mysql-connector-java-5.1.14-bin.jar:jstar-1.0.jar:jstar_test.jar com.ysnky.jstartest.JStarTest 2003 127.0.0.1 2000
.................ObserverNotifier.....................
======================Jstar Server v1.0======================
.................SystemVar start up.....................
Server starts to listening at port: 2003
sending... ip:127.0.0.1, port: 2000, cmd: [[register][2003]]
Message: [[ok.][5]]
Connection closed...
New Client Accepted: 127.0.0.1
inputLine: [[stop]]
cmd: stop
Connection closed...
New Client Accepted: 127.0.0.1
inputLine: [[start][4][1][5,127.0.0.1,2003,1,1307830835739][1,127.0.0.1,2000,0,1307830555785][3,127.0.0.1,2001,1,1307830829816]
cmd: start
new message
observer size:1
Connection closed...
sending... ip:127.0.0.1, port: 2000, cmd: [[ping_client][5]]
Message: [[ok.]]
Connection closed...
sending... ip:127.0.0.1, port: 2000, cmd: [[ping_client][5]]
Message: [[ok.]]
Connection closed...
sending... ip:127.0.0.1, port: 2000, cmd: [[ping_client][5]]
Message: [[ok.]]
Connection closed...
```

Şekil 5. 8 Kümeye 3.düğümün eklenmesi

Kümeye 3.düğüm eklendikten sonra alınan küme listesi Şekil 5.9'da verilmiştir.

```
# telnet 127.0.0.1 2000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[[get_list]]

----------------------------------------NODE----------------------------------------
id:5
ip:127.0.0.1
port:2003
type:slave
mode:1/4
lastupdate:1307830877754
------------------------------------------------------------------------------------


----------------------------------------NODE----------------------------------------
id:1
ip:127.0.0.1
port:2000
type:master
mode:0/4
lastupdate:1307830555785
------------------------------------------------------------------------------------


----------------------------------------NODE----------------------------------------
id:3
ip:127.0.0.1
port:2001
type:slave
mode:2/4
lastupdate:1307830871836
------------------------------------------------------------------------------------


----------------------------------------NODE----------------------------------------
id:4
ip:127.0.0.1
port:2002
type:slave
mode:3/4
lastupdate:1307830875750
------------------------------------------------------------------------------------
```

Şekil 5. 9 Kümeye 3. düğümün eklenmesinden sonra alınan küme listesi

Bir düğüm yöneticisi ve 3 adet düğümden oluşan kümeden bir düğümün düşürülmesi ve belirli bir süre beklendikten sonra (kümenin yeniden yapılandırılısı için gerekli olan süre kadar) küme listesinin alınması işlemi Şekil 5.10'da verilmiştir.

```
# ps -ef | grep java
  501 60631 55205  com.ysnky.jstartest.JStarTest 2000
  501 60734 55323  com.ysnky.jstartest.JStarTest 2001 127.0.0.1 2000
  501 60864 55666  grep java
  501 60774 58454  com.ysnky.jstartest.JStarTest 2002 127.0.0.1 2000
  501 60807 58519  com.ysnky.jstartest.JStarTest 2003 127.0.0.1 2000
# kill -9 60774
# telnet 127.0.0.1 2000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[[get_list]]

--------------------------------------NODE--------------------------------------
id:5
ip:127.0.0.1
port:2003
type:slave
mode:1/3
lastupdate:1307831003789
--------------------------------------------------------------------------------


--------------------------------------NODE--------------------------------------
id:1
ip:127.0.0.1
port:2000
type:master
mode:0/3
lastupdate:1307830555785
--------------------------------------------------------------------------------


--------------------------------------NODE--------------------------------------
id:3
ip:127.0.0.1
port:2001
type:slave
mode:2/3
lastupdate:1307831004891
--------------------------------------------------------------------------------
```

Şekil 5. 10 Kümeden bir düğümün çıkartılması ve yeni küme listesinin alınması

Şekil 5.11'de kümeden düğüm yöneticisinin düşürülmesi ve belirli bir süre sonra küme listesinin alınması gösterilmiştir. Burada dikkat edilecek olursa liste 2001 kapısından hizmet veren yeni küme düğüm yöneticisinden alınmıştır. Çünkü 2000 kapısından hizmet veren eski düğüm yöneticisi artık yoktur.

```
# ps -ef | grep java
  501 60631 55205  com.ysnky.jstartest.JStarTest 2000
  501 60734 55323  com.ysnky.jstartest.JStarTest 2001 127.0.0.1 2000
  501 60864 55666  grep java
  501 60807 58519  com.ysnky.jstartest.JStarTest 2003 127.0.0.1 2000
# kill -9 60631
# telnet 127.0.0.1 2001
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[[get_list]]

--------------------------------------NODE--------------------------------------
id:5
ip:127.0.0.1
port:2003
type:slave
mode:1/2
lastupdate:1307831108821
--------------------------------------------------------------------------------


--------------------------------------NODE--------------------------------------
id:3
ip:127.0.0.1
port:2001
type:master
mode:0/2
lastupdate:1307831067905
--------------------------------------------------------------------------------
```

Şekil 5. 11 Kümeden düğüm yöneticisinin çıkartılması ve yeni küme listesi

Şekil 5.12'de küme düğüm yöneticisinin düşürülmesi sonucunda 3 id'li düğümde yaşanan durum verilmiştir.

```
sending... ip:127.0.0.1, port: 2000, cmd: [[ping_client][3]]
Message: [[ok.]]
Connection closed...
java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:333)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:195)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:182)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:432)
        at java.net.Socket.connect(Socket.java:525)
        at java.net.Socket.connect(Socket.java:475)
        at java.net.Socket.<init>(Socket.java:372)
        at java.net.Socket.<init>(Socket.java:186)
        at tr.edu.yildiz.jstar.core.utils.NetworkUtil.sendMessage(NetworkUtil.java:32)
        at tr.edu.yildiz.jstar.core.utils.NetworkUtil.sendMessage(NetworkUtil.java:16)
        at tr.edu.yildiz.jstar.core.network.NodeInformer.slaveCycle(NodeInformer.java:105)
        at tr.edu.yildiz.jstar.core.network.NodeInformer.run(NodeInformer.java:57)
Connection closed...
ping failed....
java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:333)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:195)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:182)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:432)
        at java.net.Socket.connect(Socket.java:525)
        at java.net.Socket.connect(Socket.java:475)
        at java.net.Socket.<init>(Socket.java:372)
        at java.net.Socket.<init>(Socket.java:186)
        at tr.edu.yildiz.jstar.core.utils.NetworkUtil.sendMessage(NetworkUtil.java:32)
        at tr.edu.yildiz.jstar.core.utils.NetworkUtil.sendMessage(NetworkUtil.java:16)
        at tr.edu.yildiz.jstar.core.network.NodeInformer.slaveCycle(NodeInformer.java:105)
        at tr.edu.yildiz.jstar.core.network.NodeInformer.run(NodeInformer.java:57)
Connection closed...
ping failed....
```

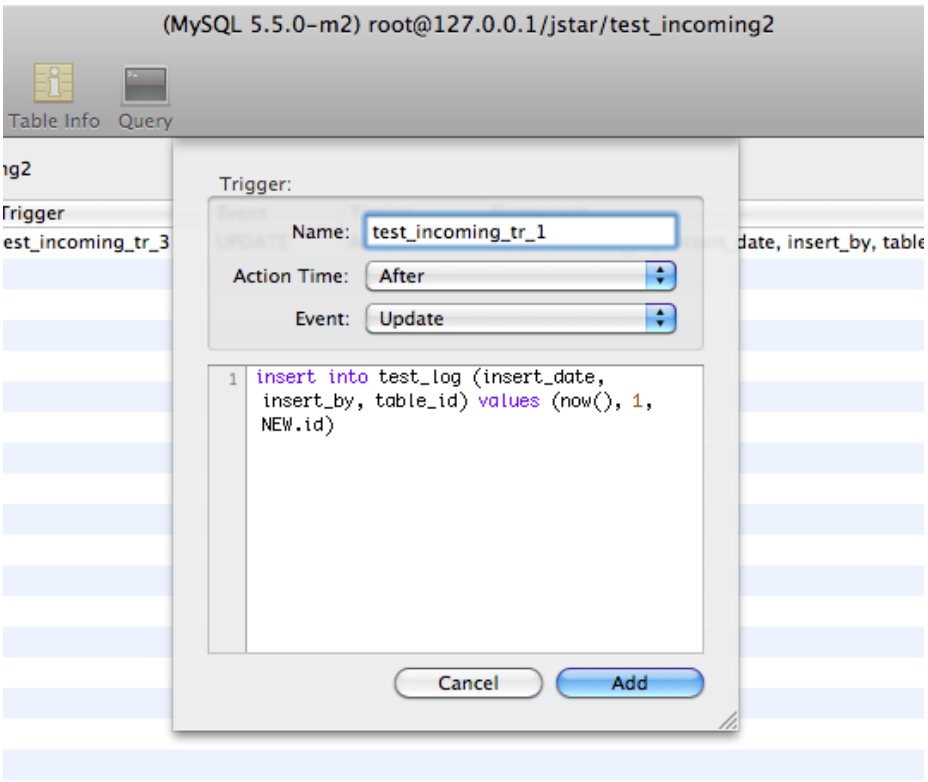
Şekil 5. 12 Düğüm yöneticisine bağlantı kuramayan bir düğüm

5.2 JStar Kümesi Kaynak Erişim Testleri

Kaynak erişiminde MySql sunucu veritabanı kullanılmış olup üzerinde çalışılacak tablo 4. Bölümde anlatılan test_incoming1 tablosudur. Bu test kapsamında test_incoming1 tablosuna sürekli kayıt girmek amacıyla farklı bir uygulama geliştirilmiştir. Geliştirilen bu uygulama ile test_incoming1 tablosuna atılan kayıtların process_count'u 0 olarak belirtilmiştir.

test_incoming1 tablosundaki process_count değeri 0 olan kayıtları alarak işleyen ve sonuçta da ilgili kaydın process_count değerini bir arttıran ve jstar-1.0.jar framework'ü üzerinden veri erişimi yapan basit bir uygulama daha geliştirilmiştir. Bu şekilde geliştirilmiş olan uygulamamız küme düzeninden çalıştırılacaktır.

Uygulamanın küme düzeninde çalıştırılması esnasında bir kaydın birden fazla işleme girmediğinden emin olmak için test_incoming1 tablosunun güncelleme olayına bir veritabanı tetikleyicisi yazılmıştır. Bu tetikleyici sayesinde ilgili kaydın id'si test_log tablosuna girilmiş olacaktır. Bu sayede test_incoming1 tablosunda gerçekleştirilecek her güncelleme işleminde test_log tablosunda bir kayıt oluşturacaktır. Ve test_log tablosundan alacağımız rapor ile de herhangi bir kaydın birden fazla işlenip işlenmediğini kontrol etme şansımız olacaktır. Oluşturulan tetikleyici Şekil 5.13'de verilmiştir.



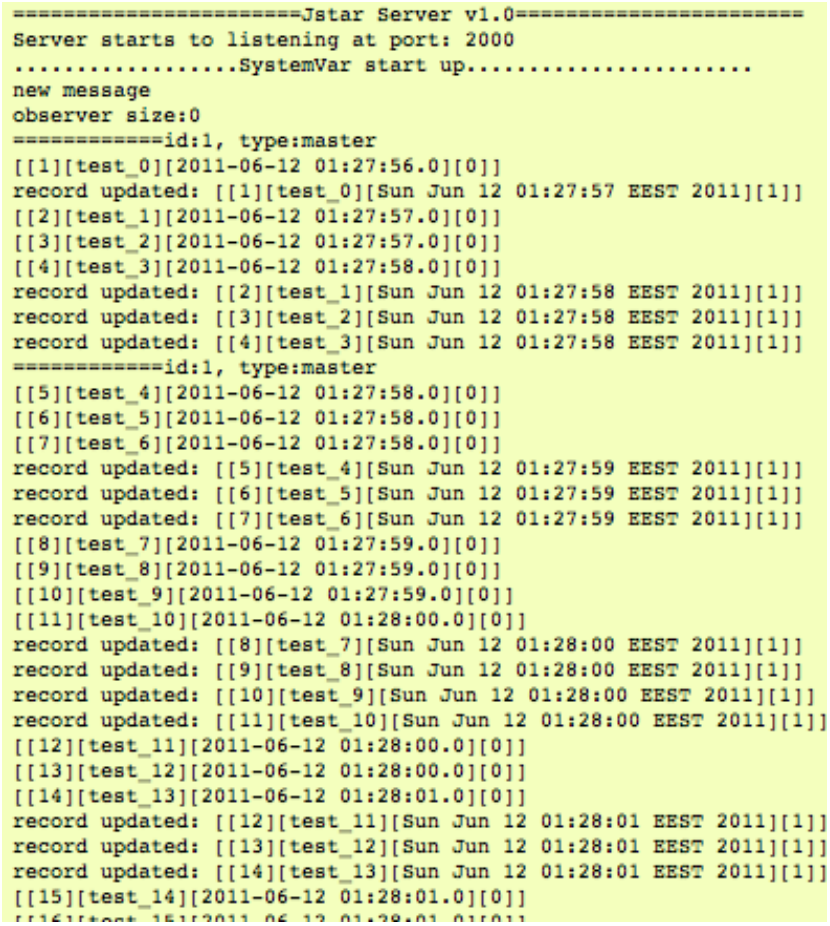
Şekil 5. 13 “test_incoming1” tablosu üzerinde tetikleyici oluşturulur

Şekil 5.14’te gösterildiği gibi incoming_test1 tablosuna kayıt girmek için jstar_test uygulaması çalıştırılmıştır.

```
# java -cp .:mysql-connector-java-5.1.14-bin.jar:jstar_test.jar com.ysnky.jstartest.DBFeeder
new record inserted: [[0][test_0][Sun Jun 12 01:27:56 EEST 2011][0]]
new record inserted: [[0][test_1][Sun Jun 12 01:27:57 EEST 2011][0]]
new record inserted: [[0][test_2][Sun Jun 12 01:27:57 EEST 2011][0]]
new record inserted: [[0][test_3][Sun Jun 12 01:27:58 EEST 2011][0]]
new record inserted: [[0][test_4][Sun Jun 12 01:27:58 EEST 2011][0]]
new record inserted: [[0][test_5][Sun Jun 12 01:27:58 EEST 2011][0]]
new record inserted: [[0][test_6][Sun Jun 12 01:27:58 EEST 2011][0]]
new record inserted: [[0][test_7][Sun Jun 12 01:27:59 EEST 2011][0]]
new record inserted: [[0][test_8][Sun Jun 12 01:27:59 EEST 2011][0]]
new record inserted: [[0][test_9][Sun Jun 12 01:27:59 EEST 2011][0]]
new record inserted: [[0][test_10][Sun Jun 12 01:28:00 EEST 2011][0]]
new record inserted: [[0][test_11][Sun Jun 12 01:28:00 EEST 2011][0]]
new record inserted: [[0][test_12][Sun Jun 12 01:28:00 EEST 2011][0]]
```

Şekil 5. 14 “test_incoming1” tablosuna kayıt giren uygulama

Şekil 5.15’te JStar çatısı üzerinden veri erişimi yapan örnek uygulamamızın ilk örneği ayağa kaldırılıyor. Bu düğüm aynı zamanda küme düğüm yöneticisidir.



Şekil 5. 15 Örnek uygulamanın ilk örneği çalıştırılıyor

Şekil 5.16'da örnek uygulamamızın ikinci örneği ayağa kaldırılmıştır. Günlükler dikkatli incelendiğinde düğümler arasında dengeli bir yük dağılımı yapıldığı rahatlıkla gözlemlenecektir.

```
# java -cp .:mysql-connector-java-5.1.14-bin.jar:jstar-1.0.jar:jstar_test.jar com.ysnky.jstartest.JStarTest 2001 127.0.0.1 2000
.................ObserverNotifier.....................
======================Jstar Server v1.0=======================
Server starts to listening at port: 2001
.................SystemVar start up.....................
sending... ip:127.0.0.1, port: 2000, cmd: [[register][2001]]
Message: [[ok.][2]]
Connection closed...
java.lang.NullPointerException
        at tr.edu.yildiz.jstar.shell.adaptor.db.JStarMySQLAdaptor.fetchData(JStarMySQLAdaptor.java:44)
        at tr.edu.yildiz.jstar.test.IncomingDAO.fetchData(IncomingDAO.java:28)
        at tr.edu.yildiz.jstar.test.DBProcessor.processData(DBProcessor.java:30)
        at tr.edu.yildiz.jstar.test.DBProcessor.run(DBProcessor.java:21)
New Client Accepted: 127.0.0.1
inputLine: [[stop]]
cmd: stop
Connection closed...
New Client Accepted: 127.0.0.1
inputLine: [[start][2][1][2,127.0.0.1,2001,1,1307831292365][1,127.0.0.1,2000,0,1307831273783]]
cmd: start
new message
observer size:1
Connection closed...
[[53][test_52][2011-06-12 01:28:13.0][0]]
[[55][test_54][2011-06-12 01:28:14.0][0]]
record updated: [[53][test_52][Sun Jun 12 01:28:14 EEST 2011][1]]
record updated: [[55][test_54][Sun Jun 12 01:28:14 EEST 2011][1]]
[[57][test_56][2011-06-12 01:28:14.0][0]]
[[59][test_58][2011-06-12 01:28:15.0][0]]
record updated: [[57][test_56][Sun Jun 12 01:28:15 EEST 2011][1]]
record updated: [[59][test_58][Sun Jun 12 01:28:15 EEST 2011][1]]
[[61][test_60][2011-06-12 01:28:15.0][0]]
[[63][test_62][2011-06-12 01:28:16.0][0]]
record updated: [[61][test_60][Sun Jun 12 01:28:16 EEST 2011][1]]
record updated: [[63][test_62][Sun Jun 12 01:28:16 EEST 2011][1]]
```

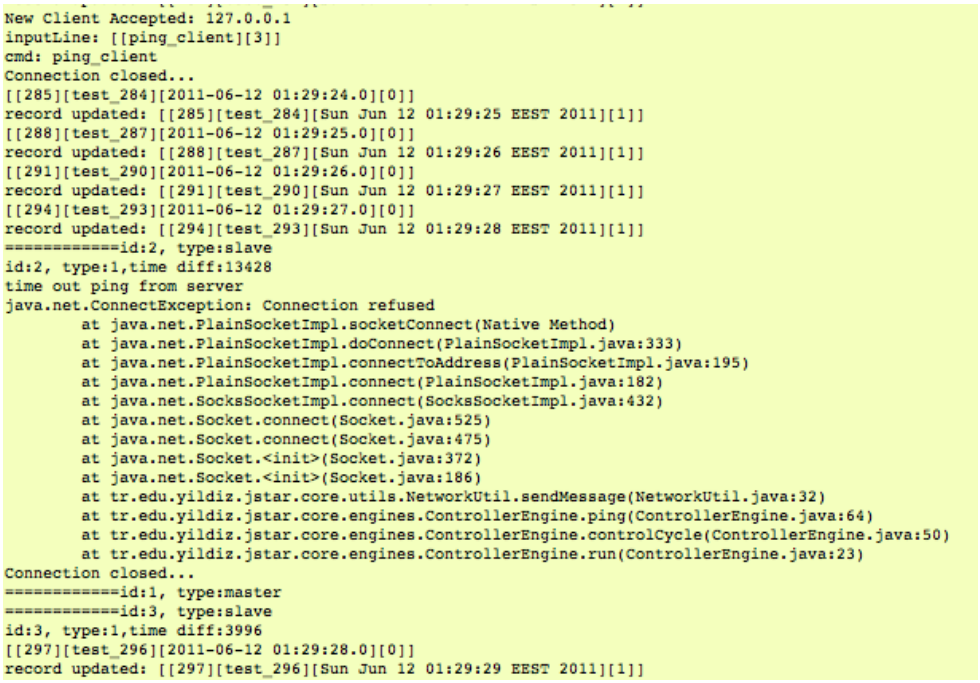
Şekil 5. 16 Örnek uygulamanın ikinci örneği çalıştırılıyor

Şekil 5.17'de örnek uygulamamızın üçüncü örneği ayağa kaldırılmıştır. Yine günlüklere bakıldığında düğümler arasında dengeli bir yük dağılımı yapıldığı gözlemlenecektir.

```
# java -cp .:mysql-connector-java-5.1.14-bin.jar:jstar-1.0.jar:jstar_test.jar com.ysnky.jstartest.JStarTest 2002 127.0.0.1 2000
..................ObserverNotifier......................
======================Jstar Server v1.0======================
Server starts to listening at port: 2002
..................SystemVar start up......................
sending... ip:127.0.0.1, port: 2000, cmd: [[register][2002]]
Message: [[ok.][3]]
Connection closed...
java.lang.NullPointerException
        at tr.edu.yildiz.jstar.shell.adaptor.db.JStarMySQLAdaptor.fetchData(JStarMySQLAdaptor.java:44)
        at tr.edu.yildiz.jstar.test.IncomingDAO.fetchData(IncomingDAO.java:28)
        at tr.edu.yildiz.jstar.test.DBProcessor.processData(DBProcessor.java:30)
        at tr.edu.yildiz.jstar.test.DBProcessor.run(DBProcessor.java:21)
java.lang.NullPointerException
        at tr.edu.yildiz.jstar.shell.adaptor.db.JStarMySQLAdaptor.fetchData(JStarMySQLAdaptor.java:44)
        at tr.edu.yildiz.jstar.test.IncomingDAO.fetchData(IncomingDAO.java:28)
        at tr.edu.yildiz.jstar.test.DBProcessor.processData(DBProcessor.java:30)
        at tr.edu.yildiz.jstar.test.DBProcessor.run(DBProcessor.java:21)
java.lang.NullPointerException
        at tr.edu.yildiz.jstar.shell.adaptor.db.JStarMySQLAdaptor.fetchData(JStarMySQLAdaptor.java:44)
        at tr.edu.yildiz.jstar.test.IncomingDAO.fetchData(IncomingDAO.java:28)
        at tr.edu.yildiz.jstar.test.DBProcessor.processData(DBProcessor.java:30)
        at tr.edu.yildiz.jstar.test.DBProcessor.run(DBProcessor.java:21)
java.lang.NullPointerException
        at tr.edu.yildiz.jstar.shell.adaptor.db.JStarMySQLAdaptor.fetchData(JStarMySQLAdaptor.java:44)
        at tr.edu.yildiz.jstar.test.IncomingDAO.fetchData(IncomingDAO.java:28)
        at tr.edu.yildiz.jstar.test.DBProcessor.processData(DBProcessor.java:30)
        at tr.edu.yildiz.jstar.test.DBProcessor.run(DBProcessor.java:21)
java.lang.NullPointerException
        at tr.edu.yildiz.jstar.shell.adaptor.db.JStarMySQLAdaptor.fetchData(JStarMySQLAdaptor.java:44)
        at tr.edu.yildiz.jstar.test.IncomingDAO.fetchData(IncomingDAO.java:28)
        at tr.edu.yildiz.jstar.test.DBProcessor.processData(DBProcessor.java:30)
        at tr.edu.yildiz.jstar.test.DBProcessor.run(DBProcessor.java:21)
New Client Accepted: 127.0.0.1
inputLine: [[stop]]
cmd: stop
Connection closed...
New Client Accepted: 127.0.0.1
inputLine: [[start][3][2][2,127.0.0.1,2001,1,1307831313372][1,127.0.0.1,2000,0,1307831273783][3,127.0.0.1,2002,1,1307831308797]]
cmd: start
new message
observer size:1
Connection closed...
[[119][test_118][2011-06-12 01:28:33.0][0]]
[[122][test_121][2011-06-12 01:28:34.0][0]]
record updated: [[119][test_118][Sun Jun 12 01:28:35 EEST 2011][1]]
record updated: [[122][test_121][Sun Jun 12 01:28:35 EEST 2011][1]]
sending... ip:127.0.0.1, port: 2000, cmd: [[ping_client][3]]
Message: [[ok.]]
```

Şekil 5. 17 Örnek uygulamanın üçüncü örneği çalıştırılıyor

İkinci örnek sonlandırılarak kümenin yeniden yapılandırılması test edilmiş ve bütün bu süreç boyunca düğüm yöneticisinde oluşan günlükler Şekil 5.18'de verilmiştir. Günlüklerden de rahatlıkla gözlemlenebileceği gibi bütün bu süreç boyunca düğümler kendi görevlerine (veri işleme) devam etmiştir.



Şekil 5. 18 İkinci örnek sonlandırıldıktan sonra düğüm yöneticisine ait günlük

İkinci örneğin sonlandırılmasından yeni kümenin oluşturulmasına kadar geçen süre boyunca her bir düğüm kendi veri işleme görevine devam etmiştir fakat ikinci örneğe ait kayıtlara dokunulmamıştır. Yeni küme oluşturulduktan sonra bu kayıtlar da küme düğümleri arasında dengeli bir şekilde paylaştırılmıştır. Şekil 5.19'da bu durum rahatlıkla gözlemlenebilir.

```
Connection closed...
new message
observer size:1
formated cmd:[[start][2][1][1,127.0.0.1,2000,0,1307831273783][3,127.0.0.1,2002,1,1307831392825]]
sending... ip:127.0.0.1, port: 2002, cmd: [[start][2][1][1,127.0.0.1,2000,0,1307831273783][3,127.0.0.1,2002,1
Message: [[ok.]]
Connection closed...
new message
observer size:1
============id:1, type:master
============id:3, type:slave
id:3, type:1,time diff:993
[[262][test_261][2011-06-12 01:29:17.0][0]]
[[268][test_267][2011-06-12 01:29:19.0][0]]
[[274][test_273][2011-06-12 01:29:21.0][0]]
[[280][test_279][2011-06-12 01:29:23.0][0]]
[[286][test_285][2011-06-12 01:29:25.0][0]]
[[292][test_291][2011-06-12 01:29:26.0][0]]
[[298][test_297][2011-06-12 01:29:28.0][0]]
[[304][test_303][2011-06-12 01:29:30.0][0]]
[[310][test_309][2011-06-12 01:29:32.0][0]]
[[316][test_315][2011-06-12 01:29:34.0][0]]
[[322][test_321][2011-06-12 01:29:36.0][0]]
[[328][test_327][2011-06-12 01:29:38.0][0]]
[[334][test_333][2011-06-12 01:29:39.0][0]]
[[340][test_339][2011-06-12 01:29:41.0][0]]
[[346][test_345][2011-06-12 01:29:43.0][0]]
[[352][test_351][2011-06-12 01:29:45.0][0]]
[[358][test_357][2011-06-12 01:29:47.0][0]]
[[364][test_363][2011-06-12 01:29:49.0][0]]
[[370][test_369][2011-06-12 01:29:50.0][0]]
[[376][test_375][2011-06-12 01:29:52.0][0]]
[[380][test_379][2011-06-12 01:29:54.0][0]]
[[382][test_381][2011-06-12 01:29:54.0][0]]
record updated: [[262][test_261][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[268][test_267][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[274][test_273][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[280][test_279][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[286][test_285][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[292][test_291][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[298][test_297][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[304][test_303][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[310][test_309][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[316][test_315][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[322][test_321][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[328][test_327][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[334][test_333][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[340][test_339][Sun Jun 12 01:29:54 EEST 2011][1]]
record updated: [[346][test_345][Sun Jun 12 01:29:54 EEST 2011][1]]
```

Şekil 5. 19 Yeni küme yapılandırılmasından sonra ikinci örneğe ait kayıtlar düğümler arasında paylaşılıyor

Tüm bu süreç boyunca oluşan kayıtlara ait günlükler test_log tablosundan Şekil 5.20'de verilen sorgu ile raporlanmıştır.

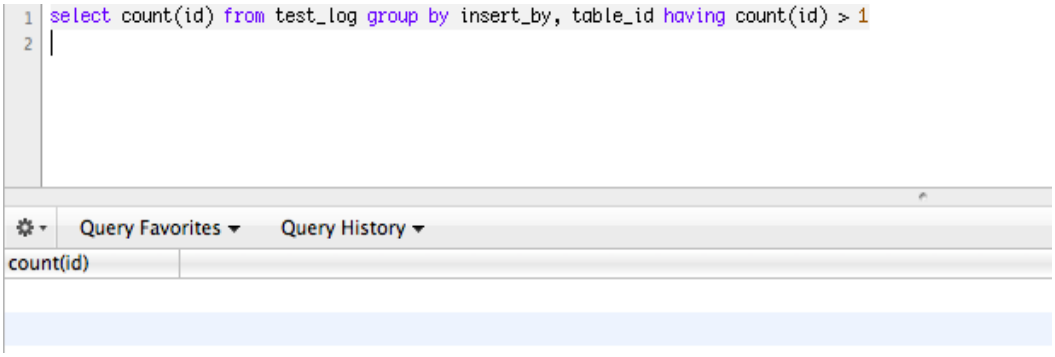
```
1 select * from test_log
2
```

Query Favorites ▾ Query History ▾

id	insert_date	insert_by	table_id
1	2011-06-11 23:30:24	1	6452
2	2011-06-11 23:30:24	1	6453
3	2011-06-11 23:30:24	1	6454
4	2011-06-11 23:30:25	1	6455
5	2011-06-11 23:30:25	1	6456
6	2011-06-11 23:30:25	1	6457
7	2011-06-11 23:30:25	1	6458
8	2011-06-11 23:30:26	1	6459
9	2011-06-11 23:30:26	1	6460
10	2011-06-11 23:30:26	1	6461
11	2011-06-11 23:30:27	1	6462
12	2011-06-11 23:30:27	1	6463

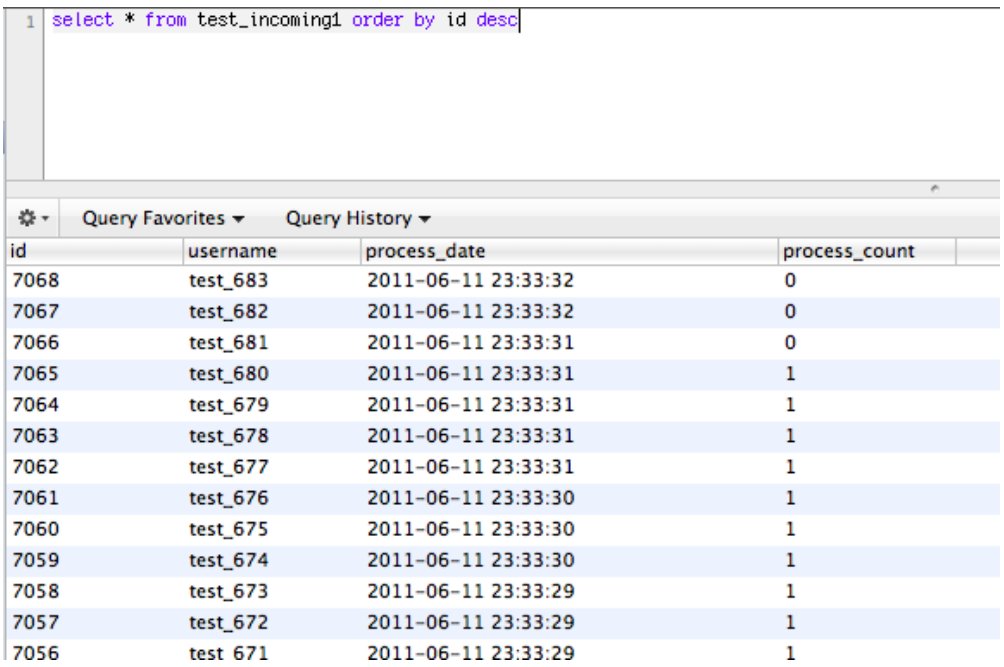
Şekil 5. 20 "test_log" tablosunda oluşan kayıtların raporlanması

Testimizin en önemli raporu Şekil 5.21'de verilmiştir. Burada yazılan sorgu cümleciği ile herhangi bir kaydın birden fazla işleme girip girmediğinin cevabı bulunmuştur. Sonuçtan da anlaşılacağı üzere birden fazla işleme giren herhangi bir kayıt bulunmamıştır. Yani küme gerek yeni bir düğüm eklenmesinde gerekse bir düğümün düşmesinde mükerrer kayıt işleme hatası yapmamıştır.



Şekil 5. 21 Bir kaydın birden fazla işlenip işlenmediğinin sorgulanması

Şekil 5.22 ise küme aktif durumda iken test_incoming1 tablosunun anlık durumunu göstermektedir. Kayıtlar kaydediliş tarihine göre tersten sıralanmıştır. Process_count değeri ilgili kaydın kaç kez işlendiği bilgisini göstermektedir. Şekilden de rahatlıkla anlaşılacağı üzere bu değer 0 (henüz işlenmemiş) ya da 1 (yalnızca bir kez işlenmiş) değerlerini almıştır.



id	username	process_date	process_count
7068	test_683	2011-06-11 23:33:32	0
7067	test_682	2011-06-11 23:33:32	0
7066	test_681	2011-06-11 23:33:31	0
7065	test_680	2011-06-11 23:33:31	1
7064	test_679	2011-06-11 23:33:31	1
7063	test_678	2011-06-11 23:33:31	1
7062	test_677	2011-06-11 23:33:31	1
7061	test_676	2011-06-11 23:33:30	1
7060	test_675	2011-06-11 23:33:30	1
7059	test_674	2011-06-11 23:33:30	1
7058	test_673	2011-06-11 23:33:29	1
7057	test_672	2011-06-11 23:33:29	1
7056	test_671	2011-06-11 23:33:29	1

Şekil 5. 22 "test_incoming1" tablosunun anlık görünümü

BÖLÜM 6

SONUÇ VE ÖNERİLER

Bu çalışma kapsamında Java programlama dili kullanılarak yine java platformuna yönelik olarak kaynak bağımlı sistemlerin küme düzeninde çalıştırılabilmesi için bir çatı geliştirilmiştir. Ayrıca geliştirilen çatı içerisinde ftp, veritabanı, dosya sistemi ve http kaynak sistemlere erişimi koylaylaştırmak ve uygulama geliştiricileri bu katmandan izole etmek için adaptörler eklenmiştir.

Geliştirilen çatı yapısı çok farklı testlere tabi tutulmuştur. Bu testler küme testleri ve kaynak erişim testleri olmak üzere iki ana başlık altında incelenmiştir.

Küme testlerinde kümenin başarılı oluşturulması, yeni düğümlerin eklenmesi, mevcut düğümlerin düşürülmesi ve düğüm yöneticisinin düşürülmesi durumları ayrıntılı olarak test edilmiş ve bütün bu durumlarda sistemin başarılı sonuçlar verdiği gözlemlenmiştir.

Kaynak erişim testlerinde ise daha çok kayıt işleme üzerinde durulmuştur. Kümeye yeni bir düğüm eklendiğinde mevcut yükün dengeli bir dağılım gösterdiği, kümeden bir düğüm çıkartıldığında yükün kalan düğümler arasında dağıtıldığı ve yine aynı şekilde düğüm yöneticisi çöktüğü durumunda da kümenin devamlılığının sağlandığı ve farklı bir düğüm yöneticisi yönteminde yeni bir yapıya geçilerek veri işlemenin devam ettirildiği gözlemlenmiştir.

Ayrıca çok kritik bir sorun olan mükerrer kayıt konusu üzerinde özellikle durulmuş ve çok farklı yöntemlerle testler yapılarak sorunun oluşturulması için çalışılmıştır fakat böyle bir durum oluşturulamamıştır. Bu durum çatı yapısının güvenilirliği adına önemli bir test sonucudur.

Geliştirilen çatı yapısının kaynak bağımlı uygulamaların küme düzeninde çalıştırılması konusunda oldukça başarılı sonuçlar verdiği gözlemlenmiştir. Geliştirilmeye açık altyapısı sayesinde çok geniş bir kullanım alanı bulacağı düşünülen JStar Java çatısı adaptörleri sayesinde de uygulama geliştiricilerin kolay adapte olmasını sağlayacaktır.

KAYNAKLAR

[1] Schroeder, T., Goddard, S. ve Ramamurthy, B., (2000). "Scalable Web Server Clustering Technologies", IEEE Network, 0890-8044.

[2] Mannell, R., A.P, A Short History of Computers and Computing, http://clas.mq.edu.au/synthesis/history_computers/index.html, 10 Haziran 2011.

[3] Java, The Java™ Tutorials, http://download.oracle.com/javase/tutorial/reallybigindex.html, 11 Haziran 2011.

[4] TIOBE, TIOBE Programming Community Index for June 2011, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, 12 Haziran 2011.

[5] Schildt, H., (2006). Java The Complete Reference, Seventh Edition, McGraw-Hill, New York.

[6] Pitt, E., (2010). Fundamental Networking in Java, First Edition, Springer, New York.

[7] Roseindia, Java Interpreter, http://www.roseindia.net/java/java-introduction/javatools/java-inetrpreter.shtml, 12 Haziran 2011.

[8] Mahovsky, J. ve Benedicenti, L., (2003). "An Architecture for Java-Based Real-Time Distributed Visualization", IEEE Transactions on Visualization and Computer Graphics, 570-579.

[9] Microsoft, Deploying Exchange Server 2003 in a Cluster, http://technet.microsoft.com/en-us/library/bb123612(EXCHG.65).aspx, 13 Haziran 2011.

[10] Chow, K. ve Kwok, Y., (2002). "On Load Balancing for Distributed Multiagent Computing", IEEE Transactions on Parallel and Distributed Systems, 787-801.

[11] Ng, K. ve Wang, W.Y.H., (2004), "Design and implementation of algorithm with multichannel load balancing and failover for generic storage area networks", Communications Systems, 2004. ICCS 2004. The Ninth International Conference on Digital Object Identifier: 10.1109/ICCS.2004.1359389

[12] Ciurana, E., High-Availability, Fault Tolerance and Resource Oriented Computing, http://ciurana.eu/GeeCON-2010, 15 Haziran 2011

[13] Vogels, W., Dumitriu, D., Agrawal, A., Chia, T. ve Guo, K., (2000). "Scalability of the Microsoft Cluster Service", 2nd USENIX Windows NT Symposium

[14] Sosinsky, B., (2011). Cloud Computing Bible, First Edition, Wiley Publishing, Indianapolis.

[15] IBM, Grid Computing Nedir?, http://www-05.ibm.com/tr/solutions/edu/grid.html, 15 Haziran 2011.

[16] Bauer, M. A. C., Erickson, N., Finnigan, D. L., Hong, P. J., Larson, J. W., Pachl, P. A., Slonim, J., Taylor, J. ve Teorey, D. J., (2010). "A distributed system architecture for a distributed application environment", IBM Systems Journal, 0018-8670

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Yasin KAYA

Doğum Tarihi ve Yeri : 01.05.1980

Yabancı Dili : İngilizce

E-posta : ysnky@yahoo.com

ÖĞRENİM DURUMU

Derece	**Alan**	**Okul/Üniversite**	**Mezuniyet Yılı**
Yüksek Lisans	Bilgisayar Müh.	Yıldız Teknik Üniversitesi	2011
Lisans	Bilgisayar Müh.	Yıldız Teknik Üniversitesi	2004
Lise	Fen	Çorum Atatürk Lisisi	1998

İŞ TECRÜBESİ

Yıl	**Firma/Kurum**	**Görevi**
2009	Turkcell Teknoloji A.Ş	Yazılım Mühendisi
2006	Avea İletişim A.Ş	Yazılım Mühendisi
2003	TOL	Yazılım Mühendisi