Küme Bilgisayarlara Görev Ataması
Task Assignment for Cluster Computers

Mustafa Haluk AKGÜNDÜZ[1]
İTÜ Bilgisayar ve Bilişim Fak.
akgunduz@itu.edu.tr

Eşref ADALI
İTÜ Bilgisayar ve Bilişim Fak.
adali@itu.edu.tr

Öz

Günümüzde başarımı yeterli sayılabilecek düşük maliyetli atanmış bilgisayarlar değişik alanlarda kullanılmaktadır. Bu makalede, bu tür atanmış bilgisayarların küme yapısında kullanılmalarına yönelik bir çalışmanın sonuçları anlatılmıştır.

Çok sayıda işlem gerektiren uygulamalarda bazı işlemler yinelenmektedir; bazı işlemlerin art arda yapılması gerekir; bazı işlemler de koşut biçimde gerçekleştirilebilir. İşlemlerin süresi değişken olabilir. Koşut işlemler aynı anda gerçekleştirilebilirler. Tek kısıt kümedeki bilgisayar sayısıdır. Art arda yapılması gereken işlemler, doğası gereği birbirini beklerler.

Bu makalede, yukarıda belirtilen koşullarda, işleri küme içindeki bilgisayarlara dağıtacak yeni bir algoritma tanıtılacak ve uygulama örnekleri sunulacaktır. Daha önceki çalışmadan [2] farklı olarak işlerin birbirleriyle olan bağımlılıklarının haritası çıkarılıp, koşut çalışma bakış açısından en iyilenerek bilgisayarlara dağıtılacaktır. Çıkarılan haritanın taranması aşamasında çeşitli senaryolar incelenecektir.

***Anahtar Sözcükler**: Atanmış bilgisayarlar, Küme bilgisayarlar, Dağıtık sistemler*

Gönderme ve kabul tarihi: 26.05.2018-20.12.2018
M. H. Akgündüz: orcid.org/0000-0001-8771-9592
E. Adalı: orcid.org/0000-0002-1561-8255
Makale türü: Araştırma

Abstract

Today's low cost and sufficient performance computers are used in different areas. In this paper the result of cluster computing application of dedicated low-cost computers are introduced.

In some of the applications that consists of many operations, some operations are repeated; some of them have to be done sequentially; some of them have to done in parallel. The operations are to be different. The parallel operation can be done at the same time interval. The only constraint is the number of computers available. The sequential operation must be done in sequence.

In this paper a new algorithm that works in conditions mentioned above is introduced and some experimental result will be given. Apart from the previous study [2], map of the dependencies between the tasks will be extracted and they will be distributed to computers with parallel process perspective. Several scenarios will be examined during the scan process of the dependency map.

***Keywords**: Dedicated computers, Cluster computers, Distributed systems*

1. Giriş

Günümüzdeki bilgisayarların büyük çoğunluğunun kullandığı mikroişlemci temelli MİB'lerin işlem hızları 1976-1986 yıllarında ortalama %25 1986-2002 yıllarında ortalama %50 ve daha sonraki yıllarda %20 oranında artmıştır [1]. Bu kaynakların da söylediği gibi işlemci hızları eskiden olduğu kadar hızlı biçimde artmamaktadır. Bunun nedeni, kırmık üretiminde molekül boyutlarına yaklaşılmış olmasıdır.

Bilgi sistemlerindeki hızlı gelişmeler, insanlar için yeni uygulama ve olanaklar sunmaktadır. Ancak insanlar daha hızlı ve daha yetenekli bilgisayarlar istemektedir. Bilgisayarın işlem hızını artırmanın bir yolu, onları koşut çalıştırmaktır. Koşut işleme konusu 1970'lerden beri süre gelen çalışma konusudur. Bir işi birden çok bilgisayara yaptırarak, o işin daha kısa sürede tamamlamak beklenen bir sonuçtur. Ancak, her zaman karşımıza şu soru çıkmaktadır:

- Her iş parçalara bölünebilir mi?
- Her iş koşut biçimde yapılabilir mi?
- Koşut ve ardışıl yapılacak işlemler küme içindeki bilgisayarlara nasıl dağıtılabilir?

Bu makalede, öncelikle yukarıdaki soruların karşılığını vermek üzere geliştirilen bir algoritma tanıtılacaktır. Ardından bu algoritmaya uygun olarak çalışan bir mimari sunulacaktır.

Koşut mimariler temel olarak sıkı ve gevşek bağlı koşut mimariler olarak iki kümeye ayrılırlar. Sıkı mimariler, bir programı koşut işlenebilecek parçalara ayırmayı hedeflerken, gevşek mimariler görevleri koşut bilgisayarlara dağıtarak çözüm üretmeye çalışırlar. Bu çalışmada gevşek koşut yapılar için görevlerin paylaştırılması üzerinde odaklanılmıştır.

Gevşek koşut çalışmalar genellikle belli işlemlerin çok sayıda yapılmasının gerektiği uygulamalarda önemli olmaktadır. Örneğin işlem türü sayısı *n* ve bu işlemlerin yapılma sayısı *m* olsun. İşlem türü sayısı *n* yaklaşık 10-20 arası olurken, işlem sayısı milyonlar mertebesinde olabilmektedir. İşlemler birbirinden bağımsız olabileceği gibi birbiri ile bağımlı olabilir. Birbirinden bağımsız işlemler için geliştirilmiş bir mimari ve algoritma önceki çalışmamızda tanıtılmıştı [2].

Son yıllarda görev ataması konusuyla alakalı çokça gelişme olmakla beraber genelde çalışmalar güvenirlik ve başarım olmak üzere iki ölçüt üzerinde yoğunlaşmaktadır. Bunlardan ilki olan güvenirlik konusunda Bannister'ın [3] yedekte kaynak tutmanın yer aldığı hata toleranslı sistemi, Hariri'nin [4] görevlerin çoklu olarak koşulması, Shatz'ın [5] maliyet işlevli algoritması ve Kartik'in [6] dallandırma ve sınırlandırma yaklaşımlı algoritması gibi çalışmalar mevcuttur. Biz de önceki çalışmamızda Bannister'ın çalışmasına benzer sisteme değiştirge olarak verilen oranda yedek kaynak tutarak (havuz yedeklemesi [7]) sistemimizin güvenirliğini artıran bir çalışma yapmıştık. Görev atamasının diğer yoğunlaştığı konu ise başarımdır. Perng'in [8] haberleşme eniyilemesi, Khandelwal'ın[9] ve Younes'in [10] çalışma maliyeti eniyilemesi ve Rudolph'un [11] yük dengelemesi vb. Gibi konular bu alandaki belli başlı çalışma konularıdır. Biz de önceki çalışmamızda Rudolph'un çalışmasına benzer yük dengelemesine ek olarak, ortak kaynak kullanım eniyilemesi, düğümlerin başarım istatistiklerinin tutularak sisteme tercih değiştirgesi olarak girilmesi ve önceliklendirme gibi başarım yöntemleri kullanmıştık. Bu makalede, önceki çalışmamız birbirinden bağımsız ve birbirine bağımlı görevlerin olması durumlarını kapsayacak şekilde genişletilecektir.

Çalışma her türlü bilgisayarlar için kullanılabilir niteliktedir. Ancak deneylerimiz ARM tabanlı ve kolayca satın alınabilen (Raspberry PI, Odroid ve Beagle Bone türevleri) atanmış bilgisayarlar üzerinde gerçekleştirilmiştir. Bilindiği gibi bu bilgisayarlar, belirli bir görevi gerçekleştirmek üzere tasarlanırlar ve sınırlı sayıda arabirime sahiptirler. İşlem güçleri, bireysel bilgisayarların yaklaşık yarısı kadardır [1]. Ancak fiyatları bireysel bilgisayarların 30-50'de biri kadardır. Güç tüketimleri düşüktür ve soğutma gerektirmezler. Bu özellikleri nedeniyle bu bilgisayarlar çok sayıda bir arada kullanılabilirler; bozulmaları durumunda yedeğiyle kolayca yer değiştirebilirler.

Çalışmada bilgisayar kümeleri mimarisi [12] temel alınmış, bilgisayarlar arası iletişim arabirimi olarak soket mimarisi temelli mesaj aktarım mimarisi kullanılmıştır. Ayrıca dağıtık mimariye uygun bir yönetim yazılımı tasarlanmıştır. Ardından bu mimari yapı üzerinde, çalışma kapsamında geliştirilen görev dağıtımı algoritmasına göre görev dağıtımı yapacak yazılım çalıştırılmış ve sonuçları değerlendirilmiştir.

2. Yönetim Sistemi

Yönetim Sistemi, bilgisayar kümeleri mimarisinde, kümelere iş dağılımını yöneten, küme içindeki iletişimi ve yedeklemeyi düzenleyen yazılımdır. Bu yazılım ayrıca bilgisayarların tamamladığı işlemlerin sonuçlarını da toplamakla yükümlüdür. Giriş bölümünde anlatıldığı gibi küme içinde bulunan bilgisayarlara dağıtılacak görevler birbirinden bağımsız olabileceği gibi, birbirine bağlı da olabilir. Bu nedenle, yönetim sisteminin görev dağılımını buna göre yapması gerekir. Çalışma kapsamında amaca uygun bir algoritma geliştirilmiştir.

Yönetim sistemi kapsamında daha önceki çalışmada da belirtildiği gibi üç farklı rol (Dağıtıcı, Toplayıcı, Düğüm) ve bu rolleri üstlenen bilgisayarlar

bulunmaktadır.

1. **Dağıtıcı** sistemdeki tüm toplayıcı ve düğümlerin varlık bilgilerini tutar ve toplayıcılardan gelen düğüm isteklerini karşılar. Sistemde yalnızca bir tane bulunur.

2. **Toplayıcı**, uygulamanın bulunduğu bilgisayarlarda çalışıp, uygulamaya ilişkin görevlerin tanımlandığı iş dosyalarını çözmekte, bağımlılık algoritmasını çalıştırıp, görev sıralarını oluşturmakta ve her bir görev için dağıtıcıdan düğüm istemektedir. Sistemde birden fazla sayıda bulunabilir.

3. **Düğümler** ise toplayıcılardan gelen iş ve işle ilgili dosyaların çalıştırılmasını sağlamakta, sonuçta oluşan çıkış dosyalarını toplayıcıların sonraki kullanımları için toplayıcılara göndermektedir.

Yönetim sisteminin donanım ile ilişkisi Şekil-1'de çizilmiştir.

2.1 Görevin tanımlanması

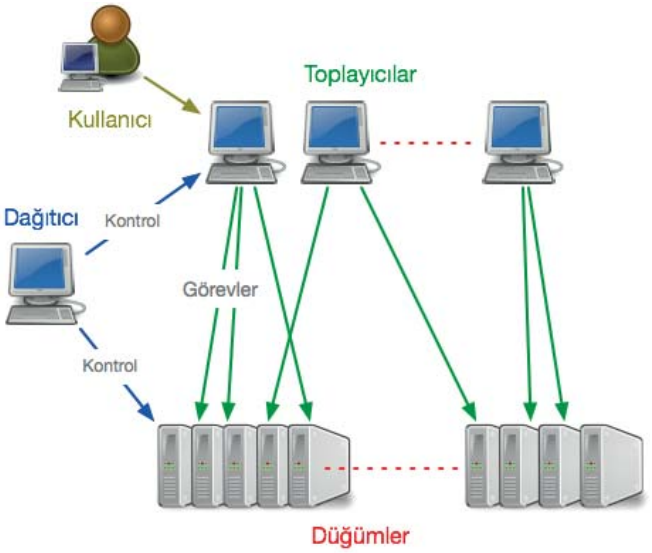
Tasarlanan sistemde görevler, kullanıcılar tarafından kurallara uygun olarak toplayıcılarda tanımlanırlar. Görevler temelde dört bileşene sahiptir;

1. **İşlevler**: Görevin düğümlerde çalıştırılmasını sağlayan uygulamalardır. Her bir görev için atanmış düğüme aktarılır. Çalışmada **"F"** ile imgelenmişlerdir.

2. **Bağımsız giriş dosyaları**: Görevlerde işlevlere değişken olarak atanan ve toplayıcıların sisteminde var olan dosyalardır. Çalışmada **"G"** ile imgelenmişlerdir.

3. **Ara giriş/çıkış dosyaları**: Görevlerde işlevlerin değişkeni ya da işlevlerin çıktısı olan ve toplayıcıların sisteminde var olmayan dosyalardır. Bu dosyalar belirli görevlerin çıktısı olarak üretilip bağımlı başka görevlere girdi oluşturmaktadır. Çalışmada **"A"** ile imgelenmişlerdir.

4. **Çıkış dosyaları**: Görevlerde işlevlerin ürettiği ve başka görevlerde kullanılmayan sonuç dosyalarıdır. Görev kümesinin hedef çıktısı olarak düşünülebilirler. Çalışmada **"Ç"** ile imgelenmişlerdir



Şekil-1: Yönetim sisteminin donanım ile ilişkisi

Kullanıcılar, toplayıcı bilgisayarlarda belirli bir işe ilişkin görev listesini Çizelge-1'deki örnekte olduğu gibi oluştururlar.

Çizelge-1: Görev Çizelgesi

İşlem sırası	Görev Listesi
1	F1 G1 G2 G6 **A3**
2	F3 A3 **A4**
3	F2 A4 A10 **Ç5**
4	F1 G6 G7 G12 **A8**
5	F2 G2 A8 **A9**
6	F1 A9 A14 **A10**
7	F3 G18 **Ç11**
8	F1 A9 G12 A17 **A13**
9	F3 A13 **A14**
10	F1 G10 G14 G18 **Ç15**
11	F2 G7 G16 G19 **A17**
12	F2 A13 A17 **A18**
13	F3 G19 **Ç20**

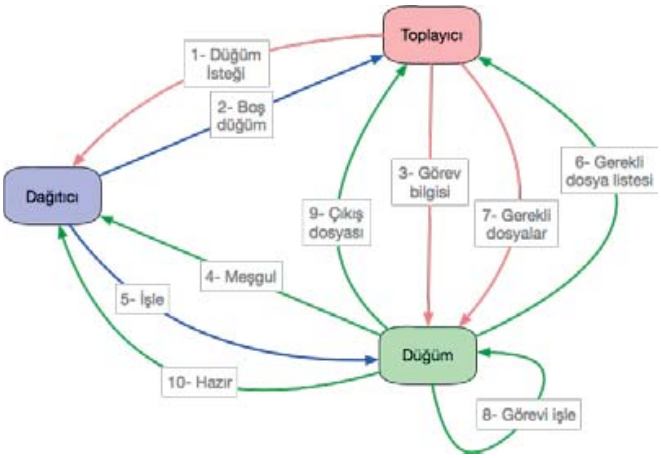
Bu örnekteki görevlerden 1, 3 ve 13. sırada olanlar aşağıda açıklanmıştır,

- "F1 G1 G2 G6 A3": F1 işlevi G1, G2 ve G6 bağımsız girdilerini kullanarak A3 ara çıktısını üretir.
- "F2 A4 A10 Ç5": F2 işlevi A4 ve A10 ara

girdisini kullanarak Ç5 hedef çıktısını üretir.

- "F3 G19 Ç20": F3 işlevi G19 bağımsız girdisini kullanarak Ç20 hedef çıktısını üretir.

Her bir görev, toplayıcılar tarafından dağıtıcının atadığı düğümlere gönderilirler. Düğümler görevi gerçekleştirdikten sonra oluşan yeni ara ya da hedef çıktıyı toplayıcıya gönderirler. Her bir görev için uygulanan akış çizgesi Şekil-2'de verilmiştir.



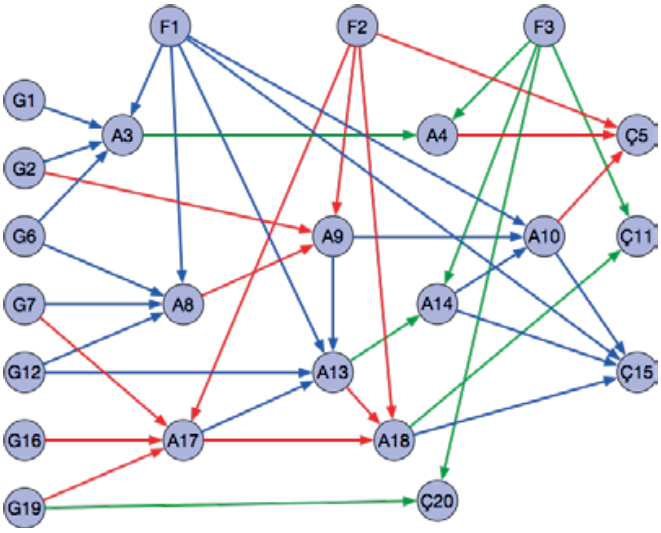
Şekil-2: Görev akış çizgesi

3. Görev Dağıtım Algoritması

Görevlerin birbiri ile ilişkileri toplayıcılarda bağlılık çizgesi biçiminde oluşturulacaktır. Şekil-3'te Çizelge-1'deki görev listesine ilişkin oluşturulmuş bağlılık çizgesi verilmiştir.

Örnek çizgede çalıştırılmak istenen görevlerde kullanılacak;

İşlevler [F1, F2, F3],
Girişler [G1, G2, G6, G7, G12, G16, G19],
Ara giriş/çıkışlar [A3, A4, A8, A9, A10, A13, A14, A17, A18] ve
Çıkışlar [Ç5, Ç11, Ç15, Ç20]



Şekil-3: Örnek bir bağlılık çizgesi

olarak imlenmiştir. Şekilden görüldüğü gibi bir çıkışın üretilebilmesi için bazı ara işlemlerin yapılması gerekmektedir. Örneğin 11 sayılı çıkışın üretilebilmesi için onbir giriş; [G2, G6, G7, G12, G16, G19, A8, A9, A13, A17, A18] katkı sağlamaktadır. Yine şekilden görüldüğü gibi bazı işlemler koşut olarak gerçekleştirilebileceği gibi bazı işlemlerin belli bir sırada yapılması gerekmektedir. Diğer taraftan işlevlerin kendi arasındaki döngüsel

İşlevlerin birbirleriyle olan bağımlılıklarının çıkartılmasında Kahn'ın [13] ilingesel algoritması iki açıdan temel alınmış ve koşut çalışmaya elverişliliği açısından incelenmiştir. Koşut çalışmaya elverişli diğer başka algoritmalar; örnek bir koşut DFS (PDFS) [14] algoritması, örnek bir koşut BFS (PBFS) algoritması [15], Blelloch'un deterministik BFS algoritması [16] ve Ligra'nın yön eniyilemeli koşut BFS algoritması [17] ve .makale kapsamında sonraki çalışmalara bırakılmıştır.

Kahn algoritmasında temel amaç düğümler arası bağımlılığı bulmak olduğu için koşutluğu sağlama gibi eniyilemeler algoritmada hedeflenmemiştir. Algoritma kendi içinde ağacı iki farklı şekilde tarayabilmektedir. İşlenecek düğümlerin listeden çekilmesi ***yığın*** ve ***kuyruk*** mantığında gerçekleştirilip sonuçları incelenmiştir.

Geliştirilen algoritmanın kurulabilmesi için aşağıda anlatılan adımlar izlenir.

1. Adım: Komşuları Belirleme

İlk aşamada tüm düğümler için komşuluk ilişkileri oluşturulur. Örneğin 1. düğümün 3. düğüm ile 2. düğümün 3 ve 9. düğümler ile komşuluk ilişkisi vardır. İşlevleri temsil eden düğümlerle birlikte tüm düğümlerin komşuluk ilişkileri Çizelge-2'de gösterilmiştir.

Çizelge-2: Komşuluk Çizelgesi

Bağ[F1] = 3, 8, 10, 13, 15	Bağ[A10] = 5, 15
Bağ[F2] = 5, 9, 17, 18	Bağ[Ç11] = Ø
Bağ[F3] = 4, 11, 14, 20	Bağ[G12] = 8, 13
Bağ[G1] = 3	Bağ[A13] = 14, 18
Bağ[G2] = 3, 9	Bağ[A14] = 10, 15
Bağ[A3] = 4	Bağ[Ç15] = Ø
Bağ[A4] = 5	Bağ[G16] = 17
Bağ[Ç5] = Ø	Bağ[A17] = 13, 18
Bağ[G6] = 3, 8	Bağ[A18] = 11, 15
Bağ[G7] = 8, 17	Bağ[G19] = 17, 20
Bağ[A8] = 9	Bağ[Ç20] = Ø
Bağ[A9] = 10, 13	

2. Adım: Derinlikleri Belirleme

Bağımlılığı olan her düğümün derinlik değerleri oluşturulur. Örneğin 13 sayılı düğüme bağlı 4 düğüm (F1, A9, G12 ve A17) bulunmaktadır. Benzer biçimde her düğüm için derinlik değeri oluşturulur. Doğal olarak giriş düğümlerinin derinlik değeri sıfırdır. Bu işlemlerin sonunda Çizelge-3'te görülen derinlik değerleri elde edilir. Derinlik değeri sıfır olanlar gösterilmez.

Çizelge-3: Derinlik Çizelgesi

Derinlik[A3] = 4	Derinlik[A10] = 3	Derinlik[A17] = 4
Derinlik[A4] = 2	Derinlik[Ç11] = 2	Derinlik[A18] = 3
Derinlik[Ç5] = 3	Derinlik[A13] = 4	Derinlik[Ç20] = 2
Derinlik[A8] = 4	Derinlik[A14] = 2	
Derinlik[A9] = 3	Derinlik[Ç15] = 4	

3.1.Adım: Kuyruk Tarama

İlk aşamada bir bağımsız düğümler listesi oluşturulur; bu listeye işlevler ve bağımsız tüm düğümler katılır. Örnekte **F1, F2, F3, G1, G2, G6, G7, G12, G16 ve G19** sayılı düğümler bağımsız düğümlerdir. Ardından sırayla her listedeki her düğüm ***kuyruk*** mantığında işlenir. Bunun için Kuyruğun ***başından*** düğüm çekilir, işlenir ve sonuç listesine eklenir. Aşağıda bu aşamalardan bazıları sırasıyla gösterilmiştir.

1 sayılı işlev düğümü için işlem adımları:

Bağımsız Liste (BL) = **F1**, F2, F3, G1, G2, G6, G7, G12, G16 ve G19
Sonuç Liste (SL) = Boş

a. Komşuluk çizelgesinden 1. işlev düğümünün bağlılığı bulunur: Bağ[F1] = 3, 8, 10, 13, 15.
b. Derinlik çizelgesinden 3. düğümün derinlik değeri 1 düşürülür. Derinlik[A3] = ~~4~~, 3 olur.
c. Derinlik çizelgesinden 8. düğümün derinlik değeri 1 düşürülür. Derinlik[A8] = ~~4~~, 3 olur.
d. Derinlik çizelgesinden 10. düğümün derinlik değeri 1 düşürülür. Derinlik[A10] = ~~3~~, 2 olur.
e. Derinlik çizelgesinden 13. düğümün derinlik değeri 1 düşürülür. Derinlik[A13] = ~~4~~, 3 olur.
f. Derinlik çizelgesinden 15. düğümün derinlik değeri 1 düşürülür. Derinlik[Ç15] = ~~4~~, 3 olur.

2 sayılı işlev düğümü için işlem adımları:

Bağımsız Liste (BL) = **F2**, F3, G1, G2, G6, G7, G12, G16 ve G19
Sonuç Liste (SL) = F1

a. Komşuluk çizelgesinden 2. işlev düğümünün bağlılığı bulunur: Bağ[F2] = 5, 9, 17, 18.
b. Derinlik çizelgesinden 5. düğümün derinlik değeri 1 düşürülür. Derinlik[Ç5] = ~~3~~, 2 olur.
c. Derinlik çizelgesinden 9. düğümün derinlik değeri 1 düşürülür. Derinlik[A9] = ~~3~~, 2 olur.
d. Derinlik çizelgesinden 17. düğümün derinlik değeri 1 düşürülür. Derinlik[A17] = ~~4~~, 3 olur.
e. Derinlik çizelgesinden 18. düğümün derinlik değeri 1 düşürülür. Derinlik[A18] = ~~3~~, 2 olur.

Bu aşamadan sonra benzer şekilde sırayla 3 sayılı işlev düğümü ile 1 ve 2 sayılı giriş düğümleri işlenir. Sonrasında 6 sayılı düğüm işlenir. Bu işlemde dikkat edilmesi gereken nokta bağımsız listenin güncellenmesi adımında listenin sonuna eklenen yeni düğümdür.

6 sayılı giriş düğümü için işlem adımları:

Bağımsız Liste (BL) = **G6**, G7, G12, G16 ve G19

Sonuç Liste (SL) = F1, F2, F3, G1, G2

a. Komşu çizelgesinden 6. giriş düğümünün bağlılığı bulunur. Bağ[G6] = 3, 8.
b. Derinlik çizelgesinden 3. düğümün derinlik değeri 1 düşürülür. Derinlik[A3] = ~~1~~, 0 olur.
c. 3. düğümün derinlik değeri 0'a düştüğü için 3. düğüm bağımsız düğüm listesinin **sonuna** eklenir.
d. Derinlik çizelgesinden 8. düğümün derinlik değeri 1 düşürülür. Derinlik[A8] = ~~3~~, 2 olur.

Düğüm işleme adımları sırasıyla 7, 12, 16, 19 sayılı giriş düğümlerinin işlenmesiyle devam eder. Bu aşamadan sonra giriş düğümlerinin işlenmesi bitmiş, sıra ara düğümlerin işlenmesine gelmiştir. İlk olarak bağımsız listenin başındaki 3 sayılı ara düğüm işlenir.

3 sayılı ara düğüm için işlem adımları:

Bağımsız Liste (BL) = **A3**, A8, A17, Ç20
Sonuç Liste (SL) = F1, F2, F3, G1, G2, G6, G7, G12, G16, G19

a. Komşu çizelgesinden 3. ara düğümün bağlılığı bulunur. Bağ[A3] = 4.
b. Derinlik çizelgesinden 4. düğümün derinlik değeri 1 düşürülür. Derinlik[A4] = ~~1~~, 0 olur.
c. 4. düğümün derinlik değeri 0'a düştüğü için 4. düğüm bağımsız düğüm listesinin **sonuna** eklenir.

Düğüm işleme adımları sırasıyla 8, 17 sayılı ara düğümlerinin işlenmesiyle devam eder. Bu aşamada sıra 20 sayılı ilk çıkış düğümüne gelmiştir.

20 sayılı çıkış düğümü için işlem adımları:

Bağımsız Liste (BL) = **Ç20**, A4, A9
Sonuç Liste (SL) = F1, F2, F3, G1, G2, G6, G7, G12, G16, G19, A3, A8, A17

a. Komşu çizelgesinden 20. düğümün çıkış düğümü olduğu görülür. Doğrudan sonuç listesine eklenir.

Benzer şekilde düğüm işleme bağımsız listede düğüm kalmayana kadar devam eder. Tarama bittiğinde aşağıdaki gibi bir işlem listesi oluşur.

Bağımsız Liste (BL) = Boş
Sonuç Liste (SL) = F1, F2, F3, G1, G2, G6, G7, G12, G16, G19, **A3, A8, A17, Ç20, A4, A9, A13, A14, A18, A10, Ç11, Ç5, Ç15**

3.2.Adım: Yığın Tarama

Kuyruk taramadaki bağımsız düğümler listesi aynı şekilde oluşturulur. Örnekte **F1, F2, F3, G1, G2, G6, G7, G12, G16 ve G19** sayılı düğümler bağımsız düğümlerdir.
Ardından sırayla her listedeki her düğüm ***yığın*** mantığında işlenir. Bunun için yığının tepesinden düğüm çekilir, işlenir ve sonuç listesine eklenir. Kuyruk taramaya benzer şekilde düğümler işlenmeye başlanır ve bağımsız listede düğüm kalmayana kadar devam eder. Tarama bittiğinde aşağıdaki gibi bir işlem listesi oluşur.

Bağımsız Liste (BL) = Boş
Sonuç Liste (SL) = G19 G16 G12 G7 G6 G2 G1 F3 **Ç20** F2 **A17** F1 **A8A9 A13 A18 Ç11 A14 A10 Ç15 A3 A4 Ç5**

Çizelge-4: Kahn Algoritmasının Etkisi

İşlem sırası	Algoritmasız	Kuyruk taramalı	Yığın taramalı
1	F1 G1 G2 G6**A3**	F1 G1 G2 G6 **A3**	F3 G19 **Ç20**
2	F3 A3 **A4**	F1 G6 G7 G12 **A8**	F2 G7 G16 G19 **A17**
3	F2 A4 A10 **Ç5**	F2 G7 G16 G19 **A17**	F1 G6 G7 G12 **A8**
4	F1 G6 G7 G12 **A8**	F3 G19 **Ç20**	F2 G2 A8 **A9**
5	F2 G2 A8 **A9**	F3 A3 **A4**	F1 A9 G12 A17 **A13**
6	F1 A9 A14 **A10**	F2 G2 A8 **A9**	F2 A13 A17 **A18**
7	F3 G18 **Ç11**	F1 A9 G12 A17 **A13**	F3 G18 **Ç11**
8	F1 A9 G12 A17 **A13**	F3 A13 **A14**	F3 A13 **A14**
9	F3 A13 **A14**	F2 A13 A17 **A18**	F1 A9 A14 **A10**
10	F1 G10 G14 G18 **Ç15**	F1 A9 A14 **A10**	F1 G10 G14 G18 **Ç15**
11	F2 G7 G16 G19 **A17**	F3 G18 **Ç11**	F1 G1 G2 G6 **A3**
12	F2 A13 A17 **A18**	F2 A4 A10 **Ç5**	F3 A3 **A4**
13	F3 G19 **Ç20**	F1 G10 G14 G18 **Ç15**	F2 A4 A10 **Ç5**

4. Sonuçlar

Görev dağıtım algoritması kullanılmadan önce ve sonrasında düğümler arasındaki bağımlılıklar ve işlem sırası Çizelge-4'teki gibi olur. Çizelgeden görüldüğü gibi yapılan işlemler aynıdır ancak yapılış sıraları farklıdır. Algoritma ardışıl işlemleri sıraya dizmektedir. Örneğin ilk işlem olarak A3'ü üretmektedir. Bu çıkışı üretmek için gerekli olan G1, G2 ve G6 girişleri bağımsız düğümlerdir. Benzer mantıkla, ikinci aşamada A8 ara çıkışını elde etmek için gerekli olan G6, G7 ve G12 düğümleri ile üçüncü

aşamada A17 ara çıkışı için gerekli olan G7, G16 ve G19 düğümleri de bağımsız düğümlerdir. Fakat A9 düğümü için gerekli olan düğümlerden G2 bağımsız,

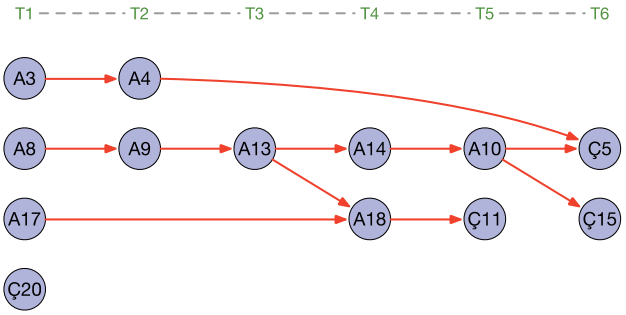
A8 ise ikinci sıradaki işlemin sonucunu beklemek durumundadır.

Daha önce açıkladığımız gibi, Kahn algoritması koşut çalışma için eniyilemeyi amaçlamamaktadır. Fakat çalışmamızda koşut çalışma görev dağıtımında önemli olduğundan algoritmanın iki farklı tarama kipi koşut çalışmaya uygunluk anlamında karşılaştırılmıştır.

Burada ilk göze çarpan durum, her bir işlemden sonra geride kalan tüm işlemler bağımsızlık durumlarına göre güncellendiğinde her iki tarama şeklinin de aynı başarıma sahip olduğu görülmüştür. Fakat her işlemden sonra tablonun tümünün güncellenmesi ek bir başarım kaybına yol açmaktadır.

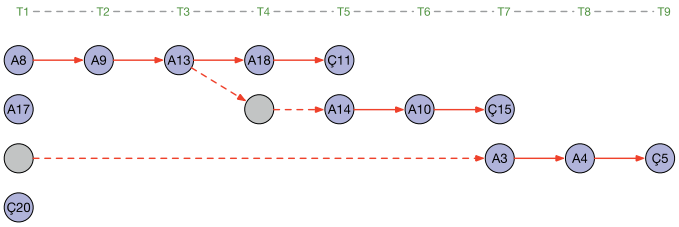
Diğer taraftan tüm tablonun taranması yerine o anki işlenen göreve en yakın bağımsız işlemlerin bulunması durumunda farklı sonuçlar elde edilmiştir. Bütün işlemlerin aynı T süresinde gerçekleştiği varsayımı altında;

Kuyruk tarama kipinde Şekil-4'teki gibi toplam T6 birimlik bir sonuç elde edilmiştir. Bu sonuç az önce bahsedilen tüm tablonun güncellenmesi durumuyla aynı zaman başarımına sahip olmakla beraber, tablo güncellemesinin getirdiği başarım kaybını barındırmamaktadır.



Şekil-4: Kahn kuyruk tarama zaman grafiği

Buna karşın yığın tarama kipinde işlemler gerçeklendiğinde ise Şekil-5'teki gibi toplam T9 birimlik bir sonuç elde edilmiştir.



Şekil-5: Kahn yığın tarama zaman grafiği

Şekil-5'ten de görüleceği gibi **A14** gereksiz yere bir T birim, **A3** ise altı T birim geç başlamıştır.

Elde edilen sonuçlara göre kuyruk tarama kipinde koşulan Kahn algoritması, yığın taramaya göre koşutsal anlamda daha iyi başarım göstermiştir.

Bir diğer taraftan standart DFS algoritması geçici bir yığın kullanarak ilingesel sıralamaya uygun hale getirilip denenmiştir. Deneme sonucu oluşan bağımlılık listesi Şekil-5'teki gibi Kahn'ın yığın taramalı kipiyle aynı sonucu vermiştir.

Sonuç olarak algoritma, koşut ve ardışıl işlemleri bağımlılıklarını göz önüne alarak sıraya dizmektedir. Yönetim yazılımı da bu algoritmadan çıkan sıraya göre görevleri maksimum koşutlukta düğümlerde çalışmasını sağlamaktadır.

Çalışmada geliştirilen algoritma ve uygulandığı mimariye ilişkin kaynak koda https://github.com/akgunduz/bankor üzerinden erişilebilir.

5. Kaynaklar

[1] *History of ProcessorPerformance*, http://www.cs.columbia.edu/~%20sedwards/classes/2012/3827-spring/advanced-arch-2011.pdf

[2] M.H. AKGUNDUZ, E. ADALI,*Atanmış Sistemler ile Bilgisayar Kümesi Tasarımı ve Mevcut Sistemler ile Karşılaştırması.* TBV, Bilgisayar Bilimleri ve Mühendisliği Dergisi, Sayı:8.2, 2015

[3] J.A. BANNISTER, K.S. TRIVEDI, *TaskAllocationin Fault-Tolerant Distributed Systems.*ActaInformatica, Vol:20, 1983

[4] S. HARIRI, C. S. RAGHAVENDRA, *Distributed FunctionsAllocationforReliabilityandDelayOptimization*,ACM '86 Proceedings of 1986 ACM Fall jointcomputerconference, 1986

[5] S.M. SHATZ, J. WANG, M. GOTO,*TaskAllocationforMaximizingReliability of Distributed ComputerSystems*IEEE Transactions on Computers Volume 41 Issue 9, September 1992

[6] S. KARTIK, C.S.R. MURTHY,*TaskAllocationAlgorithmsforMaximizingReliability of Distributed Computing Systems,*IEEE Transactions on Computers Volume 46 Issue 6, 1997

[7] E. ADALI, *Dağıtılmış Bilgisayarlarla Denetim Doçentlik Tezi*, İSTANBUL, TR. 1980

[8] P.R. MA, E.Y.S. LEE,M. TSUCHIYA, *A TaskAllocation Model for Distributed Computing Systems,*IEEE Transactions on Computers Volume 31 Issue 1, 1982

[9] A. KHANDELWAL, *Optimal ExecutionCost of Distributed Systemthrough Clustering*, International Journal of EngineeringScienceandTechnology, Vol: 3, issue: 3, 2011.

[10] A. YOUNES, *TaskAllocationforMinimizingCost of Distributed Computing Systems Using GeneticAlgorithms*, International Journal of Advanced Research in ComputerScienceand Software Engineering, vol: 2, issue: 9, 2012

[11] L.Rudolph,M.S. Allalouf, E.Upfal, *A Simple LoadBalancingSchemeforTaskAllocationin ParallelMachines*, SPAA '91 Proceedings of thethirdannual ACM symposium on Parallelalgorithmsandarchitectures, 1991

[12] G. PFISTER, *Cluster Computing, Encyclopedia of ComputerScience,* John WileyandSons Ltd., Chichester, UK, 2000

[13] A.B. KAHN, *TopologicalSortingof Large Networks*, Communications of the ACM, 5 (11), 1962

[14] U. A. ACAR, A. CHARGUÉRAUD, M. RAINEY, *A Work-EfficientAlgorithmforParallelUnordered Depth-First Search,* SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, TX, 2015

[15] C. E. LEISERSON, T. B. SCHARDL. *A Work-EfficientParallelBreadth-First SearchAlgorithm*, SPAA ’10, ACM,New York, NY, USA, 2010

[16] G. E. BLELLOCH, J. T. FINEMAN, P. B. GIBBONS, J. SHUN.*Internallydeterministicparallelalgorithms can be fast*. PPoPP ’12,ACM, New York, NY, USA, 2012

[17] J. SHUN and G. E. BLELLOCH. Ligra: *A LightweightGraphProcessing Framework ForShared Memory*, PPOPP ’13,ACM, New York, NY, USA, 2013