İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ÇOK İŞLEMCİLİ GERÇEK ZAMANLI SİSTEMLER İÇİN SANAL İHMAL EDİLEBİLİRLİK GÜDÜMLÜ İŞ SIRALAMA

YÜKSEK LİSANS TEZİ

Gökhan SEÇİNTİ

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

TEMMUZ 2012

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ÇOK İŞLEMCİLİ GERÇEK ZAMANLI SİSTEMLER İÇİN SANAL İHMAL EDİLEBİLİRLİK GÜDÜMLÜ İŞ SIRALAMA

YÜKSEK LİSANS TEZİ

Gökhan SEÇİNTİ
(504091513)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Yrd. Doç. Dr. D. Turgay ALTILAR

TEMMUZ 2012

İTÜ, Fen Bilimleri Enstitüsü'nün 504091513 numaralı Yüksek Lisans Öğrencisi **Gökhan SEÇİNTİ**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"ÇOK İŞLEMCİLİ GERÇEK ZAMANLI SİSTEMLER İÇİN SANAL İHMAL EDİLEBİLİRLİK GÜDÜMLÜ İŞ SIRALAMA"** başlıklı tezini aşağıdaki imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Yrd. Doç. Dr. D. Turgay ALTILAR** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Prof. Dr. Çoşkun SÖNMEZ** Yıldız Teknik Üniversitesi	.....
	Yrd. Doç. Dr. Erdem MATOĞLU İstanbul Teknik Üniversitesi	.....
		.....

Teslim Tarihi : **4 Mayıs 2012**
Savunma Tarihi : **12 Temmuz 2012**

ÖNSÖZ

Yüksek lisans eğitimim boyunca bana destek olan aileme ve arkadaşlarıma, tez sürecimde bana yol gösteren danışmanım Yrd. Doç. Dr. D. Turgay ALTILAR'a teşekkür ederim.

Temmuz 2012

Gökhan SEÇİNTİ
(Bilgisayar Mühendisi)

İÇİNDEKİLER

Sayfa

ÖNSÖZ ..... **v**
İÇİNDEKİLER ..... **vii**
KISALTMALAR ..... **ix**
ÇİZELGE LİSTESİ ..... **xi**
ŞEKİL LİSTESİ ..... **xiii**
ÖZET ..... **xv**
SUMMARY ..... **xvii**
1. GİRİŞ **1**
1.1 Tezin Amacı 2
1.2 Kuramsal Bilgiler 2
1.2.1 Gerçek zamanlı sistemlerde iş sıralama algoritmaları 3
1.2.1.1 Oransal monoton 3
1.2.1.2 En yakın zaman sınırı önce 4
1.2.1.3 En az ihmal edilebilir önce 4
2. İLGİLİ ÇALIŞMALAR **5**
2.1 Gerçek Zamanlı İş Sıralama Algoritmalarının Sınıflandırılması 5
2.2 Global ve Dinamik Öncelikli Algoritmalar 6
2.2.1 Pfair ve türevleri 6
2.2.2 LLREF ve türevleri 8
3. GELİŞTİRİLEN YÖNTEM **11**
3.1 Amaç 11
3.2 Sanal İhmal Edilebilirlik Güdümlü İş Sıralaması 11
3.2.1 Kesinti kısıtlı en az ihmal edilebilir önce 12
3.2.2 Alt görev kümesi oluşturmak 14
3.3 Doğruluk Tanıtı 18
4. GERÇEKLEŞTİRİM **21**
4.1 Simulasyon Ortamı 21
4.1.1 Görev ve kaynak modelleri 21
4.1.2 İş sıralayıcı modeli 23
4.2 Önerilen Algoritmanın Gerçekleştirimi 24
4.2.1 Görev listeleri 25
4.3 Önerilen Algoritmanın Simulasyona Eklenmesi 26
5. DENEYSEL SONUÇLAR **29**
5.1 Gerçeklenen Algoritmalar 29
5.2 Test Senaryoları 29

5.2.1 En kötü durum senaryoları 29
5.2.2 Daha iyi iş sıralaması sağlanan senaryolar 31
5.3 Sonuçlar 31
6. SONUÇ VE ÖNERİLER 33
KAYNAKLAR 35
EKLER 37
ÖZGEÇMİŞ 39

KISALTMALAR

RM : Rate Monotonic
EDF : Earliest Deadline First
LLF : Least Laxity First
BF : Boundary Fairness
VLDS : Virtual Laxity Driven Scheduling
GTS : Global Task Set
HP : High Priority
LP : Low Priority

ÇİZELGE LİSTESİ

Sayfa

Çizelge 3.1: Görev Kümesi I 13
Çizelge 3.2: CreateWorkSet fonksiyonu sonucunda bölünmüş iş örnekleri 16
Çizelge 5.1: En kötü durum senaryosu için görev kümesi 30
Çizelge 5.2: İyileştirme sağlanan durum senaryosu için görev kümesi..... 31
Çizelge A.1: Terimler Sözlüğü 38

ŞEKİL LİSTESİ

Sayfa

Şekil 3.1 : LLF (a) ve LLFPC (b) algoritmalarının Görev Kümesi I için oluşturdukları iş sıralama planı 13
Şekil 3.2 : Kesinti kısıtlı en az ihmal edilebilir önce algoritması durum geçiş şeması 13
Şekil 3.3 : Alt Görev Kümesi Zaman Şeması 15
Şekil 3.4 : VLDS algoritmasında işlerin durum geçiş şeması 17
Şekil 3.5 : VLDS akış diyagramı 18
Şekil 3.6 : Akıcı iş sıralamada çalışma süreleri dağıtımı (a: Adil dağıtım, b: Yer değiştirme sonrası) 20
Şekil 4.1 : STORM benzetim programında görevlerin durum geçiş diyagramı [1] 23
Şekil 4.2 : VLDS algoritmasında görevlerin durum geçiş diyagramı 25
Şekil 5.1 : En kötü durum senaryosuna ait görev kümesi için oluşturulan iş sıralama planı 30
Şekil 5.2 : İyileştirilmiş senaryoya ait görev kümesi için oluşturulan iş sıralama planı (a: VLDS algoritması, b: BF algoritması) 32

ÇOK İŞLEMCİLİ GERÇEK ZAMANLI SİSTEMLER İÇİN SANAL İHMAL EDİLEBİLİRLİK GÜDÜMLÜ İŞ SIRALAMA

ÖZET

Bu tezde çok işlemcili sistemler için gerçek zamanlı bir iş sıralama algoritması sunulmuştur. Çalışmada kullanılan sistemin eş işlemciler içerdiği ve görev kümesinin ise yalnızca örtülü zaman sınırlarına sahip periyodik görevlerden oluştuğu kabul edilmiştir. Bu konuda yapılan güncel çalışmalarda, iş sıralama algoritmaları adil davranma prensibi üzerine yoğunlaşmış olmasına rağmen önerilen algoritmada bu kısıtlara bağlı kalmadan başarımı daha yüksek iş planları üretilmesi amaçlanmıştır.

Tezde önerilen, Sanal İhmal Edilebilirlik Güdümlü (VLDS) iş sıralama algoritmasında adil davranma sınırları göz ardı edilerek bağlam değişimi sayısı düşürülmüş ve işlerin ortalama tepki süreleri iyileştirilimiştir. Bununla beraber VLDS algoritması yalnızca adil davranarak çözümlenebilecek görev kümeleri için de iş planlaması yapabilmektedir.

VLDS algoritması iki ana parçadan meydana gelmektedir. Bunlardan ilki iş sıralama süreci boyunca mevcut görev kümesini belirli zaman aralıkları için alt kümelere ayıran algoritmayı içeren kısım, ikincisi ise bu alt kümeler içerisinde iş sıralamayı gerçekleştiren kısımdır.

VLDS algoritmasın alt görev kümelerinin oluşturulması en yakın zaman sınırına D_n göre yapılmaktadır. Amaç iş sıralaması yapılacak olan işlerin aynı zaman sınırına sahip olmasına sağlayabilmektir. Bu sağlandıktan sonra LLF (en az ihmal edilebilir önce) gibi algoritmalarla çözülebilmektedir. Bu çalışmada LLF iş sıralama algoritmasında oluşabilecek yüksek sayıda bağlam değişimini engellemek için LLF algoritmasına kesinti kısıtı eklenmiştir. Kesinti kısıtlı en az ihmal edilebilr önce (LLFPC) algoritması ile oluşturulan iş planlarındaki bağlam değişim sayısı düşürülmüş ve iş sıralama algoritmasının her kuantumda çalışması gerekliliği ortadan kaldırılmıştır.

LLFPC'den önce ilk yapılması gereken sistemde bekleyen işlerden bir alt küme oluşturulmasıdır. Öncelikle en yakın zaman sınırı D_n belirlenir ve bu zaman sınırı ile belirlenen aralık için alt iş kümesi oluşturulmaya başlanır. Alt küme oluşturma işleminde tüm işler sahip oldukları zaman sınırına göre iki kümeye ayrılmaktadır. Bu kümelerden ilki Yüksek Öncelikli (HP), zaman sınırı D_n ile aynı olan işleri içeren görev kümesi; ikincisi Düşük Öncelikli (LP), zaman sınırı D_n'den yüksek olan işleri içeren görev kümesidir.

Oluşturulacak alt iş kümesinde zaman sınırlarının kaçırılmaması için HP'de bulunan tüm işlerin bu alt kümeye dahil edilmesi gerekmektedir. Zaman sınırı daha yüksek olduğu halde bu aralık içerisinde zaman sınırını kaçırma riski taşıyan işler LP kümesi içerisinde bulunabilir. Bu yüzden alt iş kümesinin oluşturulduğu zaman aralığının genişliğinden daha düşük ihmal edilebilirlik değerine ($(D_n - t) > l$) sahip

LP görevlerin zaman sınırlarını kaçırmamaları için en az ($D_n - t - l$) kadar çalışma süresi kadar işlemciye sahip olmalıdırlar. Bu atama da gerçekleştirildikten sonra boş kalan işlemci süresi LP işler arasında ihmal edilebilirlik değerine göre dağıtılmaktadır. Böylece alt iş kümesinin oluşturulması tamamlanmaktadır. En yakın zaman sınırına (D_n) gelindiğinde bu işlem tekrar yapılarak yeni bir alt iş kümesi oluşturulmaktadır.

VLDS algoritması, oluşturulan alt iş kümeleri içerisinde kullanılan LLFPC sayesinde işlerin yürütülmesi sırasında oluşabilecek kesilmelerinin sayısını azalttığından ve zaman sınırını kaçırma riski bulunmayan LP işler arasında, adil davranma kısıtı gözetmeksizin dağıtım yapabildiğinden oluşturduğu iş planlarındaki bağlam değişim sayısını düşürmekte ve işlerin ortalama tepki sürelerini azaltmaktadır.

VIRTUAL LAXITY DRIVEN SCHEDULING
FOR MULTIPROCESSOR REAL TIME SYSTEMS

SUMMARY

An optimal real time scheduling algorithm has been presented in this thesis for multiprocessor systems. It has been assumed that system consists m identical processors and only contains periodic tasks with implicit deadlines. Despite of the recent studies focus on the notion of fairness, the proposed algorithm improves average response time of the tasks and decreases the number of the context switches by reducing the fragmented execution of the tasks without any fairness constraints.

In spite of the common usage of traditional scheduling algorithms such as Earliest Deadline First (EDF) and Least Laxity First (LLF) in single processor systems, it has been shown that they are not suitable for multiprocessor scheduling problems.

In this thesis, a new algorithm, Virtual Laxity Driven Scheduling (VLDS), is proposed. The VLDS algorithm ignores fairness for the sake of both fewer context switches and shorter response times. However, the VLDS algorithm also schedules the task sets that can only be scheduled by a fair schedulers.

The VLDS is composed of two algorithms that would work consequently. One of the algorithms is used to create sub-work sets from GTS, another one is used to schedule these sub-work sets.

Basic principal of the VLDS algorithm is to create sub work sets according to the nearest deadline. All tasks in this sub-work set must share the same deadline. Then the scheduling problem can be resolved over the sub-work set with Least Laxity First(LLF). But the number of context switches increases dramatically even for a small scale systems when there exists tasks which have equal laxity values. A simple derivation of the LLF algorithm, which constraints preemption operations, has been proposed in order to reduce the number of context switches in the scheduling.

With the introduction of preemption constraints to Least Laxity First (LLF), the number of context switches and the overhead of the scheduling is reduced by removing the necessity of the laxity calculations on each quantum.

LLFPC algorithm assigns least laxity tasks first and defines next preemption point according to laxity value of the unassigned task with the least laxity. This prevents context switches until an unassigned has no (zero) laxity. When a preemption occurs, algorithm reassigns tasks by their new laxity values and define a expected new preemption time.

First step in the VLDS algorithm while creating sub-work-set is determining the nearest deadline (D_n). The duration between the current time and the D_n is called interval and and all calculations are bounded with this interval. Once the D_n is

determined, the total number of time slots provided by the processors, which is contained by the interval, is calculated.

Tasks with deadline D_n are scheduled within the interval. However, there might be the additional time slots available within the interval for use of other tasks. To distribute these time slots among the tasks, they are grouped as High Priority (HP) and Low Priority (LP) task sets.

The deadlines of the HP tasks equal to the D_n and their executions have to completed in the current interval. Therefore remaining execution time of each HP tasks must be included in the created work-set to meet deadlines.

After the HP tasks, included in the created work-set, remaining idle time is distributed among the LP tasks. LP tasks that have lower laxities than the interval could miss their deadlines if the tasks have not scheduled in the current interval. In order to prevent negative laxity (deadline miss), at least $(D_n - l_{T_i})$ time slots must be assigned for each LP task with laxity lower than the nearest deadline.

After these assignments have been made, there have to be additional remaining idle time to assign LP tasks. It has been shown with the proof of optimality that it is impossible to schedule given task set if there is no remaining idle time just after necessary distribution for LP and HP tasks to prevent deadline misses in the current interval has been made.

After the necessary distributions, remaining idle time is distributed among LP tasks according to their laxity values. Until no idle time is left in the current interval or no waiting task is left in GTS, maximum idle time that could be assigned to a LP task has been assigned by starting the LP task with the lowest laxity. This last distribution provides improvements that algorithm offers. Idle time slots left after the necessary distribution is assigned to LP tasks without any fairness constraints and allows to reduce fragmented execution of the tasks i.e, decrease the number of context switches.

Once all possible distribution of the time slots in the interval among tasks has been performed, LP Tasks which have been assigned the current work set could be considered as virtually divided into two parts. The first part of the task contains the information that is going to be used in the scheduling of the current sub work set by LLFPC algorithm. Second part holds the information that belongs to the remaining part of the task. Assigning time slots to the tasks could be considered as dividing task into two parts and assigned values will define virtual values of that task.

In current work-set tasks could be distinguished from each other under two different categories. In first category, tasks have the states of LLFPC algorithm, a task could be in any state such as Ready, Virtual Zero Laxity and Running at any time during the interval which the current work set is created for. Second category depends on the real deadline value of the task. In this category, tasks are belong to the High Priority State or Low Priority State and there is not going to be any transition between these states until the end of interval. This category has no effect on decisions of the LLFPC algorithm.

If there is a condition that the total number of time slots required by the Low Priority tasks is equal or greater than the idle time left in the interval after assignment of the High Priorty Tasks has done (totalTimeSlotsRequiredforLPTask $\geq$ idleTime) then the

scheduler could not assign the required time to LP tasks. Missing deadlines will be unavoidable. With this assumption, it will be shown that where this condition exists, total utilization of GTS is greater than the number of processors.

STORM (Simulation TOol for Real time Multiprocessor scheduling) has been used to develop and test algorithms. STORM provides discrete time simulation environment for multiprocessor systems that may contain periodic and aperiodic tasks.

A work conserving and optimal scheduling algorithm, VLDS, has been presented and it has been stated that VLDS when compared fair schedule algorithms decreases the number of context switches and improves the average response times of the tasks. This paper shows that fairness could be relaxed or even ignored in some cases to provide better schedules. As future work, cache awareness could be implemented to reduce context switches and migrations at the junctions that one interval finishes and begins. Thus, arbitrary assignments of the tasks to the processors could be replaced with assignment decisions that take into account the cache contents of the processors.

1. GİRİŞ

Günümüzde işlemci teknolojisinin işlem hızında yaşadığı engel teknolojinin çok çekirdekli işlemcilere kaymasına sebebiyet vermiştir. Böylece bilgisayar bilimlerinde birden fazla işlemcinin veya birden fazla çekirdek içeren işlemcilerin verimli bir şekilde paralel çalışmalarını sağlamak için yapılan çalışmalar hız kazanmıştır.

Tüm bilgisayar sistemlerinde olduğu gibi gerçek zamanlı sistemlerde de çok işlemcili veya çok çekirdekli işlemciler tercih edilmeye başlanmıştır. Böylece işlemciler de paralel kullanılması gereken kaynaklar halini almış ve gerçek zamanlı sistemler için uygulanabilir yeni yöntemlere ihtiyaç ortaya çıkmıştır.

Alan Burns et. al. [2]'da tanımladığı üzere Gerçek Zamanlı Sistemler, sisteminin doğruluğunun yalnızca sistem tarafından doğru çıktılar üretilmesine değil, aynı zamanda bu çıktıları ürettiği zamana da bağlı olduğu sistemlerdir. Görevlerin zaman sınırları içerisinde tamamlanacağını garanti eden ve bu zaman sınırlarının kaçırılması tolere edilemediği sistemler katı gerçek zamanlı sistemler olarak adlandırılmaktadır. Bu çalışmanın ilerleyen kısımlarında gerçek zamanlı olarak adı geçen tüm sistemler katı gerçek zamanlı sistemlerdir.

Çizelgeleme (Scheduling), kısıtlı kaynakların görevler arasında nasıl paylaşılacağının planlanmasıdır [3]. Bu çalışmada yapılan çizelgeleme algoritmasında kullanılan tek kaynak işlemciler ve kaynak talebinde tek görevler işlemciler üzerinde yürütülecek olan işlerdir. Bundan dolayı çalışmanın ilerleyen kısmında çizelgeleme yerine iş sıralama tümcesi kullanılmıştır.

Gerçek zamanlı sistemlerin tanımında da belirtildiği gibi bu sistemlerde yapılan işlerin berlirlenmiş zaman aralıklarında tamamlanması sistemin doğruluğu açısından önem taşımaktadır. Bu yüzden bu sistemlerde yer alacak farklı iş sıralama algoritmalarına ihtiyaca sebebiyet vermektedir. Geleneksel işletim sistemlerinde bulunan iş sıralama algoritmaları görev arasında adil kaynak paylaşımını amaçlarken, gerçek zamanlı

sistemlerinde kullanılan çizelgeleme algoritmalarında öncelikli amaç işleri zaman sınırları (deadline) içersinden tamamlayabilmektedir.

1.1 Tezin Amacı

Bu çalışmada çok işlemciden oluşan gerçek zamanlı sistemler için bir iş sıralama algoritması geliştirilmesi amaçlanmıştır. Geliştirilen algoritmanın literatürdeki örneklerine göre daha iyi sonuç vermesi hedeflenmiştir. çalışmada iş sıralama algoritmaları karşılaştırılırken bu algoritmaların ürettikleri iş planlarındaki bağlam değişim sayısı ve işlerin ortalama tepki süreleri göz önünde bulundurulmuştur. Yapılan çalışmada karşılaştırılan iş sıralama algoritmaları bu iki değer üzerinden kıyaslanmıştır. Bu iki değerin düşük olması sistemdeki iş sıralama maliyetini düşüreceğinden, bu değerleri daha düşük olan iş sıralama algoritmalarını daha verimli olduğu kabul edilmiştir.

1.2 Kuramsal Bilgiler

Gerçek zamanlı sistemlerde iş sıralama algoritmalarının başarılı bir çalışma sergileye-bilmesi ancak işlerin zaman sınırlarından önce tamamlanabilmesi ile mümkündür. Bundan dolayı tüm algoritmaların ilk amacı bu işlerin zaman sınırlarından önce tamamlanabilmesini garanti altına almaktır.

Geleneksel iş sıralama algoritmaları ve bunların uygulanabilirlikleri ile ilgili bilgiler verilmeden önce bu algoritmaların gerçeklendiği ortam modellerinin incelenmesi gerekmektedir.

Katı gerçek zaman ortamında işlere ait yapılan varsayımlar aşağıda verilmiştir [4].

- Zaman kısıtına sahip tüm iş istekleri periyodiktir ve istekler sabit aralıklar ile yapılmaktadırlar.
- İşler birbirinden bağımsızdır. Bir işin başlangıcı veya bitişi farklı iş isteklerine bağlı değildir.
- Bir göreve ait işin çalışma süresi o göreve ait her iş için sabittir, zamana bağlı değişim gösteremez.

1.2.1 Gerçek zamanlı sistemlerde iş sıralama algoritmaları

Bu bölümde tek işlemcili gerçek zamanlı sistemlerde kullanılan geleneksel iş sıralama algoritmalarına yer verilmiştir. Bu algoritmaların arasında ki temel farklılık yürütme sırasında işlerin birbirine olan önceliklerinin dinamik değişimi olarak nitelendirilebilinir. Önceliklerin değişimine göre iş sıralama algoritmaları üç sınıfa ayrılmaktadır. Bunlar sırasıyla statik öncelikli, iş düzeyinde dinamik öncelikli algoritmalar ve tam dinamik öncelikli algoritmalar olarak adlandırılabilir. [5] Statik öncelikli algoritmalarda bir göreve ve bu göreve ait işlerin tamamı iş sıralama işlemi boyunca aynı öncelik değerine sahiptir. İş düzeyinde dinamik öncelikli algoritmalarda ise bir göreve ait iş yürütme süresi boyunca aynı öncelik değerine sahip olmaktadır. Ama aynı göreve ait işlerin öncelikleri istek zamanlarındaki durumlara göre farklı değerler olabilmektedir. Tam dinamik algoritmalarda ise işlerin yürütme sırasında da öncelik değerleri değişebilmektedir.

Gerçek zamanlı sistemlerde yaygın olarak kullanılan üç temel iş sıralama algoritmasına kısaca açıklanmıştır. Bu iş sıralama algoritmaları öncelik değişimlerine göre farklı sınıflarda yer almaktadırlar.

1.2.1.1 Oransal monoton

Oransal monoton (Rate Monotonic - RM) iş sıralama algoritmasında, görevlere periyotlarına göre sabit öncelikler atamaktadır. Periyodu en düşük olan göreve en yüksek öncelik atanmaktadır. Bir göreve ait tüm işler yürütme süreleri boyunca aynı öncelik değerine sahip olmaktadırlar. RM iş sıralama algoritması iş yükü aşağıdaki denklem ile belirlenmiş olan görev kümeleri için başarılı bir gerçek zamanlı çizelgeleme sağlayabilmektedir [4].

$$U = \sum_{i=0}^{n} \frac{e_i}{d_i} \leqslant n \cdot \left(2^{\frac{1}{n}} - 1\right) \qquad \textbf{(1.1)}$$

Bir görevin sistemde oluşturacağı yük o görevin bir periyot içerisindeki çalışma süresi ve periyodunun oranı $\frac{e_i}{d_i}$ olarak tanımlanmaktadır. Denklem **1.1**'te verilen n

sistemde bulunan toplam görev sayısını temsil etmektedir. Denklem *n* sayısı için RM algoritmasıyla iş sıralaması yapılabilecek maksimum iş yükünü hesaplamaktadır.

1.2.1.2 En yakın zaman sınırı önce

En yakın zaman sınırı önce (EDF - Earliest Deadline First) iş sıralama algoritmasında görevler sabit bir öncelik değerine sahip değildirler. EDF algoritmasında en yakın zaman sınırına sahip işler daha yüksek önceliklere sahip olmaktadırlar. Bu yüzden bir göreve ait işlerin diğer işlere olan öncelik değerleri değişim gösterebilmektedir. EDF algoritması tek işlemcili sistemler için başarılı çizelgeleme sağlabilmekte fakat çok işlemcili sistemlerde sağlayamamaktadır.

1.2.1.3 En az ihmal edilebilir önce

Bir iş için ihmal edilebilirlik (laxity) değeri denklem **1.2**'te gibi tanımlanmaktadır.

$$laxity = absoluteDeadline - remainingExecutionTime - currentTime \qquad (\mathbf{1.2})$$

En az ihmal edilebilir önce (LLF - Least Laxity First) [6] algoritmasında ihmal edilebilirlik değeri en az olan iş en yüksek önceliği sahip olmaktadır. Bu algoritmada aynı göreve ait işler farklı önceliklere sahip olabilirken, bir işin çalışma esnasında da öncelik değeri değişim gösterebilmektedir. LLF algoritması, EDF gibi tek işlemcili sistemlerde başarılı çizelgeleme sağlayabilirken, çok işlemcili sistemler her görev kümesi için başarılı bir çizelgeleme üretememektedir.

Çok işlemcili gerçek zamanlı sistemlerde bir iş aynı anda birden fazla işlemcide işlenemeyeceği için tek işlemcili sistemler için geliştirilen iş sıralama algoritmalarının bu sistemlerde %100 yüklü iş kümeleri için çözüm üretmesi mümkün değildir. %100 yüklü iş kümelerinin çok işlemcili sistemlerde çizelgelenebilmesi için farklı yaklaşımlara ihtiyaç duyulmuştur. Bu konuda yapılmış çalışmalar ile ilgili detaylı bilgi bir sonraki bölümde verilmiştir.

2. İLGİLİ ÇALIŞMALAR

2.1 Gerçek Zamanlı İş Sıralama Algoritmalarının Sınıflandırılması

2011 yılında, katı gerçek zamanlı çok işlemcili iş sıralama algoritmaları konusunda yapılan araştırmada [7] iş sıralama algoritmaları belirli özelliklerine göre alt sınıflara ayrılmıştır. Bu sınıflandırmada da [5]'da olduğu gibi işler önceliklerine göre statik öncelikli, iş düzeyinde dinamik öncelikli ve tam dinamik öncelikli olacak şekilde üç alt sınıfa bölünmektedir. Bu ayrım dışında işlerin çalışma süreleri içerisinde işlemci değiştirmelerine (migration) göre de iş sıralama algoritmaları üç ayrı sınıfa bölünmektedir. İlk sınıf bir görev ve o göreve ait işlerin iş sıralama boyunca hangi işlemcide yer alacağının belli olduğu işlemci değiştirmenin yasak olduğu (no migration) iş sıralama algoritmaları içermektedir. Bu algoritmalar görevleri işlemcilere paylaştırarak (partitioning) iş sıralama işlemini gerçekleştirmektediler. İkinci sınıf, bir göreve ait bir işin başladığı işlemcide bitmesi zorunluluğuna sahip algoritmaları (iş düzeyinde dinamik yer değiştirme) kapsamaktadır. Bu algoritmalarda bir göreve ait iki iş farklı işlemcilerde yer alabilir fakat bir iş çalışma süresi esnasında kesilip başka bir işlemciye taşınmaz. Üçüncü sınıfa ait algoritmalarda ise işlemci değiştirme ile ilgili herhangi bir kısıtlama (fully dynamic migration) bulunmamaktadır. İşler çalışma süreleri içerisinde herhangi bir anda bir işlemci üzerindeki çalışmasını durdurup aynı anda veya daha sonra başka bir işlemcide yürütülmeye devam edebilmektedir.

Bu kategorizasyonlar altında incelendiğine bu çalışmada geliştirilen algoritma dinamik öncelikli ve dinamik işlemci değiştirebilme özelliklerine sahiptir. Bu özelliklere sahip algoritmalar sınıf Global ve Dinamik Öncelikli Algoritmalar olarak adlandırılmaktadır.

Bu sınıfa ait çalışmalar açıklanmadan önce paylaştırılmış ve global algoritmaların özelliklerini birleştiren yarı paylaşılmış melez bir algoritma ile ilgili bilgi verilmiştir.

EKG [8] (EDF with task splitting and k processors in a group) periyodik görevler ait işleri iki alt işe bölmekte ve bu alt iş parçalarını farklı zamanlarda çalışmaları için

farklı işlemcilere atamaktadır. Bu atama algoritmanını dinamik işlemci değişiklik yeteneğini tamamen ortadan kaldırmasa da kısıtlamakta ve melez (hybrid) olarak değerlendirilmesine sebebiyet vermektedir. EKG algoritmasında işlerin bölünmesi bir k değerine göre yapılmaktadır. Bu algoritmada süper periyot içersinde yapılacak olan bağlam değişim sayısı $2k$ ile kısıtlana bilmekte ancak k'nın düşük tutulması algoritmanın başarılı çalışabileceği en yüksek iş yükü oranını da düşürmektedir.

2.2 Global ve Dinamik Öncelikli Algoritmalar

Bu bölümde, geliştirilen çalışmanın içinde yer aldığı sınıfa ait diğer iş sıralama algoritmaları ile ilgili bilgi verilmiştir.

2.2.1 Pfair ve türevleri

Çok işlemcili gerçek zamanlı iş sıralama için ilk çözüm Baruah et. al. tarafından geliştirilmiştir [9]. Pfair olarak adlandırılan algoritma görevlere yüklerine göre adil kaynak atama ilkesini benimsemiştir. Her görevin yükü o görevin zaman sınırı ve o zaman sınırı içerisindeki çalışma süresinin bölümü olarak hesaplanmaktadır. Akıcı çizelgeleme (fluid scheduling) [10] yöntemindeki felsefe ayrık zamanlı çalışma ortamına aktarılmıştır. Pfair algoritması %100 yüklü görev kümeleri için başarılı bir iş sıralama sağlayabilmektedir.

Pfair algoritmasında işler bir gecikme (lag) değerine göre işlemciye sahip olmaktadırlar.

$$-1 < lag(j,t) < 1 \qquad \textbf{(2.1)}$$

Pfair, denklem **2.1**'te verilen gecikme değerini her iş için sürekli aralıkta tutmaktadır. Böylece tüm görevler yüklerine her kuantumda yüklerine göre atama yapmaktadır ve adil davranma kısıtını her kuantum için katı bir şekilde uygulamaktadır.

Bu gecikme değeri pozitif ise iş geride (behind), negatif ise ileride (ahead), sıfır ise zamanında (punctual) olarak değerlendirilmektedir. Bir işin gecikmesi ve o saat darbesiyle yapılması gereken iş miktarındaki değişimi o işin o kuantum içerisinde işlenmesi gerektiğini veya kesinlikle işlenmemesi gerektiği bilgisini verir.

Gecikme değeri dışında her iş için tutulan alfa katarı da işin bir sonraki kuantumda yapılması gerekecek olan çalışma zamanı hakkında bilgi vermektedir. Alfa katarı +,-,0 değerlerinden oluşmaktadır. Alfa katarındaki değerler yapılması gereken iş miktarındaki değişime göre hesaplanmaktadır. Pfair iş sıralama algoritması ayrık zamanda çalıştığından dolayı iş yükünün zamanla çarpımından elde edilen yapılması gereken iş miktarında ondalıklı değerler göz ardı edilmektedir. Bu değerlerin değişimi iş sıralamaya etkisi alfa katarının algoritma içerisinde oluşturulup kullanılmasıyla sağlanmaktadır.

Eğer bir iş t anında geride ve alfa katarındaki işareti eksi ise bu iş urgent (acil) işler grubuna girmektedir. Tersi şekilde iş t anında ileride ve alfa katarındaki değeri artı ise iş tnegru (işlenmemesi gereken) işler grubuna girmektedir. Bu grublardan herhangi birisine dahil olmayan işler contending (çekişen) iş grubuna alınmaktadır.

Bu grublar bir işin kuantum içerisinde işlemciye sahip olup olmaması gerektiği ile ilgili bilgi vermektedir. İşlenmesi gereken (urgent) veya işlenmemesi gereken (tnegru) iş kümelerinde dahil olmayan işler arasında bir karşılaştırma algoritmasına göre işlemciye atanacak işler seçilmektedir.

Görevlerin toplam ağırlığının işlemci sayısına eşit olduğu (%100 yüklü) görev kümeleri düşünülürse her işin zaman sınırı içerisinde tamamlanabilmesi için (çizelgelemenin başarılı olması için) kesinlikle hiç bir işlemcinin bir kuantum bile boş kalmaması gerekmektedir. Pfair algoritması her kuantum başında işlenmesi gereken işleri işlemcilere atamakta ve boş kalan işlemciler için iki gruba da (işlenmesi gereken işler, kesinlikle işlenmemesi gereken işler) dahil olmayan işleri atamaktadır.

PDfair [11] algoritması Pfair algoritmasının bazı eklentiler ile çalışma süresi iyileştirilmiş bir türevidir. Bu algoritmada contending (çekişen) işlerin tutulduğu liste içerisinde doğrusal bir sıra tanımlanmış ve böylece karşılaştırma algoritmasının çalışma süresi sabit zamanlı bir hale getirilmiştir.

Bunun dışında algoritma içerisinde tüm işler ağır (heavy), hafif (light) işler olmak üzere iki farklı gruba ayrılmıştır. İş yükü 0.5’ten büyük olan işler ağır, küçük olan işler hafif olarak değerlendirilmiştir. Algoritma içerisinde hafif işlerin ne zaman

işlemciye sahip olacağı aranırken, ağır işlerin ne zaman işlemciye sahip olmayacağı aranmaktadır. Böylece algoritmanın çalışma sayısı iyileştirilmiştir.

ERFair (Early Release Fair) [12] algoritmasında ise denklem **2.2**'te verilen işler için gecikme aralığı esnetilmiştir. İşlerin belirlenen çalışma süresinden geri kalması halen engellenirken ileride çalışma kısıtları kaldırılmıştır.

$$lag(j,t) < 1 \qquad \textbf{(2.2)}$$

Adil davranma kısıtında sağlanan bu esneklik toplam iş yükünün işlemci sayısından daha düşük olduğu görev kümelerinde (%100'ün altında yüke sahip) bağlam değişimi sayısını düşürmekte ve görevlerin tamamlanma sürelerini (response time) kısaltmaktadır.

BF (Boundary Fair) [13] algoritmasında ise Pfair algoritmasından farklı olarak adil davranma kısıtları her kuantum için uygulanmamaktadır. Yalnızca görevlerin periyotlarında (zaman sınırlarında) adil davranama kısıtları uygunlanmaktadır. Böylece görevlerin periyotları arasında adil davranma kısıtılarını sağlayabilmek için yapılan bağlam değişimlerinin oluşturduğu yükü azaltmak amaçlanmıştır. Ayrıca Pfair algoritmasının aksine her kuantumda iş sıralama algoritmasının yürütülmesi ihtiyacı da ortadan kaldırılmıştır.

2.2.2 LLREF ve türevleri

LLREF [14] (largest local remaining execution time first), işlerin zaman sınırları ve başlangıçları ile sınırlanmış aralıklar içerisinde akıcı çizelgeleme uygulamaktadır. Tanımlanan bu aralıklar Zaman ve Yerel Çalışma Zamanı Düzlemi Soyutlaması (TLPA - Time and Local Execution Time Domain Plane Abstraction) olarak adlandırılmaktadır. LLREF algoritmasında bir işin aralık içersinde yapması gereken çalışma süresi o aralığın genişliği ile o işin yükünün çarpımı ile bulunmaktadır. Aralık içersinde en yüksek yerel çalışma süresine sahip olan iş önce işlenmektedir. İş sıralama algoritması işlemciye sahip bir işin yerel çalışma süresi bittiğinde veya bir iş sıfır ihmal edilebilirlik durumuna girdiğinde çalışmakta ve işlemciye atanması gereken işi atamaktadır. Yerel aralıkta kaynakları işlerin yüklerine göre atadığından zaman sınırlar arasında görev arasında adil davranma kısıtlarını sağlamakta yani PFair algoritması ve

türevleri ile benzerlik göstermektedir. Fakat LLREF algoritması farklı olarak ayrık zamanda değil, sürekli zamanda çalışmaktadır.

LLREF algoritmasını bir türevi olan Geliştirilmiş Zaman ve Düğümsel Çalışma Zamanı Düzlemi Soyutlaması [15] (E-TNPA - Extended-Time and Nodal Execution Time Domain Plane Abstraction) algoritması farklı olarak yerel iş sıralamalarda tüm işlerin yerel zamanları tamamlanıp boşta işlemci kaldığı durumlarda işlemlerin yerel çalışma sürelerini arttırabilmektedir. %100 yüklü görev kümelerinde aynı iş planını üreten E-TNPA algoritması, daha düşük yüklü görev kümelerinde bağlam değişim sayısını ve tepki süresini düşürebilmektedir. Ayrıca sistemde bekleyen işler olduğu durumda işlemcilerin boş kalmasına izin vermediği için iş odaklı (work conserving) bir iş sıralama sağlamaktadır.

Bu bölümde verilen kısa araştırmanın da gösterdiği gibi, katı gerçek zamanlı sistemlerde örtülü zaman sınırına sahip periyodik görevlerin iş sıralaması ile ilgili yöntemler ve başarılı çözümler literatürde yıllardır mevcuttur. Güncel çalışmaların çoğu görev arasında adil davranma prensibini bensimesine rağmen bu çalışma herhangi bir adil davranma kısıtı içermeyen ve %100 yüklü görev kümeleri için daha iyi iş planları üreten bir algoritma geliştirilmiştir.

3. GELİŞTİRİLEN YÖNTEM

Yapılan çalışmada gerçek zamanlı çok işlemcili sistemler için adil davranma kısıtları bulunmayan yeni bir iş sıralama algoritması geliştirilmiştir.

Sistemde bulunan tüm görevleri içeren küme (Global Task Set), denklem **3.1** 'de tanımlanmıştır. Algoritma çalışması sırasında belirlenen zaman aralıkları için sistemde bulunan görev kümesinden alt görev kümeleri oluşturmakta ve bu zaman aralıkları içerisinde tanımlanan bu kümelerde bulunan görevler arasında iş sıralama işlemini gerçekleştirmektedir.

$$T_i \in GTS, \quad i = 1,2...n \tag{3.1}$$

3.1 Amaç

Geliştirilen Sanal İhmal Edilebilirlik Güdümlü İş Sıralama (VLDS - Virtual Laxity Driven Scheduling) algoritması adil davranma kısıtlarına sahip olmadığından işlerin daha az bağlam değişimi ile çalışmasına olanak sağlamaktadır. Adil davranma kısıtlarının göz ardı edilmesiyle zaman sınırı sistemdeki diğer görevlere göre daha uzak olan işlerin çalışmaları, yakın zaman sınırına sahip olan işler için ertelenebilmektedir. Böylece sistemde bulunan görevler için ortalama tepki süreleri düşürülebilmektedir.

3.2 Sanal İhmal Edilebilirlik Güdümlü İş Sıralaması

VLDS algoritması iki temel kısımdan oluşmaktadır. Bunlardan ilki en yakın zaman sınırına göre GTS kümesinden bir alt küme oluşturmaktır. Oluşturulan alt kümede dikkat edilmesi gereken ilk unsur zaman sınırı en yakın zaman sınırına (deadline) eşit olan tüm görevlerin kalan iş sürelerinin bu alt kümeye dahil edilmesidir. Bundan sonraki adımlarda boş (idle) işlemci sürelerinin diğer görevlerin kalan iş sürelerine paylaşımı yapılmaktadır. Diğer görevler içersinde bu dağıtım gerçeklenirken bu

görevlerin kalan iş süresi ve zaman sınırı değerleri için yeni sanal değişkenler tanımlanmaktadır. Burada tanımlanan yeni sanal değişkenler ile bu alt görev kümesi içerisinde yer alan tüm görevlerin zaman sınırlarının aynı olması sağlanmaktadır.

Alt görev kümesi oluşturulduktan sonra bu küme içerisinde iş sıralaması yapılmaktadır. Zaman sınırları aynı olan görevlerden oluşan bu kümeler içerisinde gerçekleştirilecek olan iş sıralaması LLF gibi geleneksel algoritmalarla çözümlenebilmektedir. Bağlam değişimi sayısını düşürmek için bu algoritmanın kesinti kısıtı eklenmiş basit bir türevi kullanılmıştır.

3.2.1 Kesinti kısıtlı en az ihmal edilebilir önce

Kesinti kısıtlı en az ihmal edilebilir önce (LLFPC - Least Laxity First with Preemption Constraints) algoritmasında LLF algoritmasından farklı olarak iki veya daha fazla görevin ihmal edilebilirlik değerleri eşit olduğu durumlarda sistemde sürekli bağlam değişimi yapılmasının engellenmesi amaçlanmıştır.

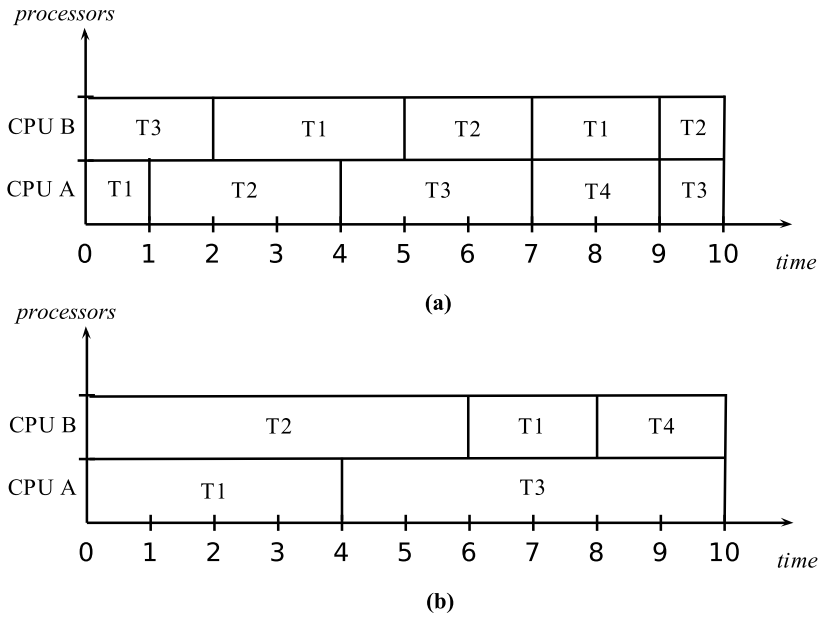
Alt görev kümelerinin tanımlandıkları aralıkların başında LLFPC ilk iş dağıtım kararlarını LLF gibi almakta ve en az ihmal edilebilirlik değerine sahip görevleri işlemcilere atamaktadır. LLFPC bundan sonra LLF'ün aksine her adımda tekrar iş sıralama kararları almamakta, işlemciye sahip olamayan görev arasında en az ihmal edilebilirlik değerine göre bir kesme noktası belirlemektedir. İlk karardan sonra LLFPC ancak bir kesme noktasına gelinirse ya da işlemciye sahip olan bir iş çalışma süresini doldurup sonlanırsa çalışmaktadır. Burada kesme noktasının ihmal edilebilirlik değerine göre belirlenmesi sistemde işlemciye sahip olamayan bir görevin ihmal edilebilirlik değerinin sıfıra gelene (Zero Laxity) kadar beklemesini sağlamaktadır. Böylece bağlam değişimi sayısı ve iş sıralama algoritmasının çalışma sayısı azaltılmaktadır.

Çizelge 3.1 verilen görev kümesi için LLF ve LLFPC algoritmalarının ürettikleri iş planları oluşturulmuş ve Şekil 3.1'te verilmiştir.

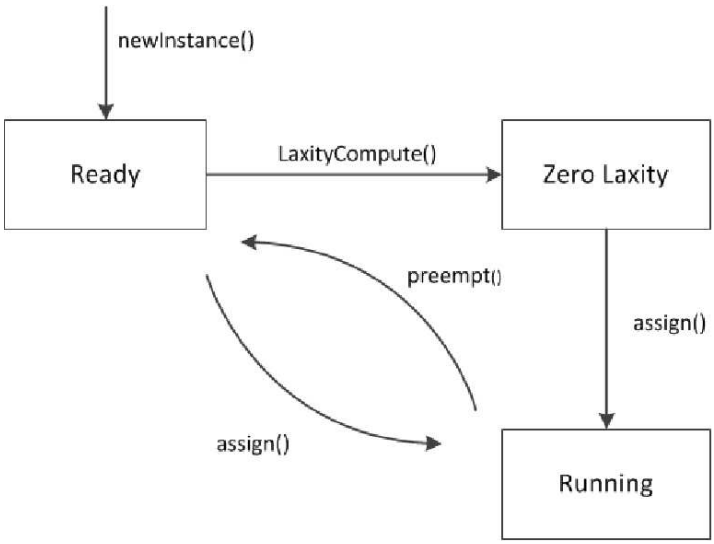
Şekil 3.1'te görüldüğü gibi eşit ihmal edilebilirlik değerleri oluştuğu durumlarda yapılacak bağlam değişimleri engellemiş ve iş sıralama esnasında yapılan toplam değişim sayısı LLFPC algoritmasıyla düşürülmüştür.

Görev Adı	Çalışma süresi	Periyot (Zaman sınırı)
T1	6	10
T2	6	10
T3	6	10
T4	2	10

Çizelge 3.1: Görev Kümesi I



Şekil 3.1: LLF (a) ve LLFPC (b) algoritmalarının Görev Kümesi I için oluşturdukları iş sıralama planı



Şekil 3.2: Kesinti kısıtlı en az ihmal edilebilir önce algoritması durum geçiş şeması

3.2.2 Alt görev kümesi oluşturmak

Sanal İhmal Edilebilirlik Güdümlü İş Sıralaması Algoritması çalışma süresi boyunca en yakın zaman sınırı için alt görev kümeleri oluşturmakta ve bu alt görev kümeleri LLFPC algoritması ile çözümlenmektedir. Alt kümenin tanımlandığı aralık sona erdiğinde, yeniden en yakın zaman sınıra belirlenmekte ve LLCPC tarafından çözümlenmesi için tekrar alt görev kümesi oluşturulmaktadır.

$$D_n = min\{d_{T_i} |\ T_i \in GTS\}\ . \tag{3.2}$$

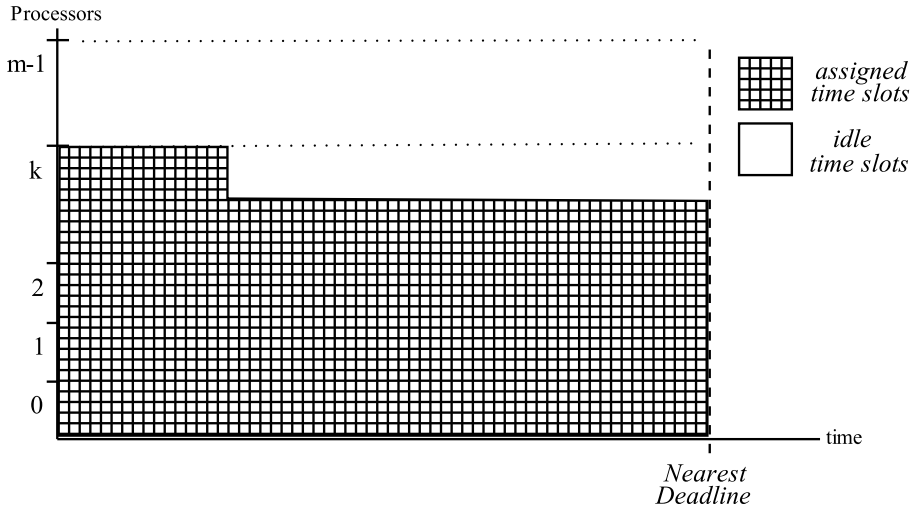
Alt kümeler oluşturulurken yukarıda da belirtildiği gibi ilk adım en yakın zaman sınırının (D_n) belirlenmesidir. En yakın zaman sırını belirlendikten sonra bu zaman sınırıyla kısıtlanan aralıkta toplam ne kadar işlemci süresi olduğu hesaplanmaktıdır. Bu hesaplama denklem **3.3**'te görüldüğü gibi alt görev kümenin tanımlandığı aralığın genişliği ile işlemci sayısının çarpımıyla gerçekleştirilmektedir. Böylece bu aralık için yaratılacak alt görev kümesinin sahip olabileceği en yüksek iş yükünün sınırı belirlenmiştir.

$$totalTimeSlots = m \cdot (D_n - t)\ . \tag{3.3}$$

Alt görev kümesin oluşturulurken mevcut görev kümesi, görevlerin zaman sınırlarına göre iki ayrı kümeye ayrılmıştır. İlk küme zaman sınırı en yakın zaman sınırı eşit olan görevleri içermekte ve Yüksek Öncelikli (HP - High Priority) görev kümesi olarak adlandırılmaktadır. Diğer küme ise zaman sınır en yakın zaman sınırından büyük olan görevleri içermekte ve Düşük Öncelikli (LP - Low Priority) görev kümesi olarak anlandırılmaktadır. Bu kümelerin tanımları denklem **3.4** ve **3.5**'te verilmiştir.

$$\begin{aligned} HP &= \{T_i |\ d_{T_i} = D_n\} \\ LP &= \{T_i |\ d_{T_i} > D_n\} \end{aligned} \tag{3.4}$$

$$HP \cup LP = GTS,\ \ HP \cap LP = \emptyset\ . \tag{3.5}$$



Şekil 3.3: Alt Görev Kümesi Zaman Şeması

HP görevler zaman sınırlarını kaçırmamak için kalan iş yüklerini (remaining execution time) tanımlanan aralıkta tamamlamak zorundadırlar. Bu görevlere atanan işlemci zaman miktarı denklem **3.6**'te verilmiştir.

$$assignedTimeSlot\,forHP = \sum_{j=0}^{|HP|} e_{T_j^{HP}} \; . \tag{3.6}$$

Bu atama gerçekleştirildikten sonra işlemciler üzerinde boş (idle) kalan zaman miktarı denklem **3.7**'teki gibi hesaplanmaktadır.

$$idleTimeSlots = m \cdot (D_n - t) - \sum_{j=0}^{|HP|} e_{T_j^{HP}} \; . \tag{3.7}$$

Alt görev kümesinin tanımlandığı aralığın genişliğinden daha düşük değerde ihmal edilebilirliğe sahip ($(D_n - t) > l_{T_i}$) LP işler tanımlanan aralık içersinde zaman sınırlarını kaçırma riskine sahiptirler. Bu görevlerin zaman sınırlarını kaçırmamasını güvence altına alabilmek için öncelikle bu görevlere $D_n - t - l_{T_i}$ kadar işlemci zamanı atanmalıdır. Böylece bu görevler sıfır ihmal seviyesinde tutularak zaman sınırlarını kaçırmaları engellenir. Tüm LP görevlerin zaman kısıtlarını kaçırmaması için gereken toplam işlemci zamanı denklem **3.8**'te olduğu hesaplanmaktadır.

$$\begin{matrix} totalTimeSlots \\ Required\,forLPTasks \end{matrix} = \sum_{k=0}^{|LP|} \left[D_n - t - l_{T_k^{LP}} \right] \; . \tag{3.8}$$

	Sanal değerler		Kalan kısma ait değerler		Gerçek değerler	
Görev adı	çalışma süresi	zaman sınırı	çalışma süresi	zaman sınırı	çalışma süresi	zaman sınırı
Task 1	3	6	2	10	5	10
Task 2	4	6	2	8	6	8

Çizelge 3.2: CreateWorkSet fonksiyonu sonucunda bölünmüş iş örnekleri

LP görevlere atanması zorunlu olan işlemci miktarları atandıktan sonra boş kalan işlemci süreleri yine LP görevler arasında ihmal edilebilirlik değerlerine göre atanmaktadır. Burada yapılan atama işlemi önerilen yöntemin literatürdeki diğer yöntemlere göre sahip olduğu avantajları sağlamaktadır. Yapılması zorunlu olan tüm atamalardan sonra kalan boş işlemci süresinin dağıtımı en düşük ihmal değerine sahip görevlere yapılabilecek en fazla atama miktarını vererek yapılmaktadır. Burada atama için üst sınır alt görev kümesinin tanımlandığı aralığın genişliği veya işin kalan çalışma süresinden küçük olanı olarak tanımlanabilir.

Yapılan bütün bu atama işlemleri aslında hangi görevin hangi işlemcide ne zamanda çalışacağını belirtmemekte yalnızca tanımlandıkları aralıkta ne kadar miktarda çalışmaları gerektiği bilgisini oluşturmaktadır. Bu bilgiler, alt görev kümesinde yer alan görevler için tanımlanan sanal zaman sınırı, sanal kalan iş yükü ve bu iki değerin farkı olan sanal ihmal edilebilirlik olarak tutulmaktadır. LLFPC algoritması tamamen bu sanal değerleri kullanarak iş sıralama işlemini gerçekleştirmektedir. Çizelge 3.2'te en yakın zaman sınır 6'ya eşit olduğu varsayılan durumda alt görev kümesinde yer alan iki görev için sanal değer örnekleri verilmiştir.

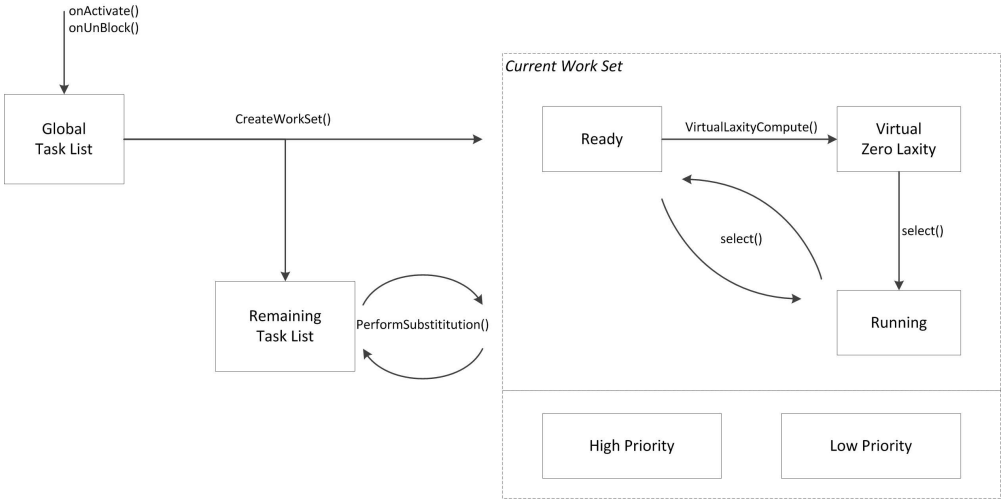
Çizelge 3.2'te de görüldüğü gibi görevlerin sanal zaman sınırları eşittir ve kullanmaları istenen işlemci süreleri de sanal kalan iş yükü değeri olarak tutulmaktadır.

Alt görev kümesi yaratma algoritmasını içeren *CreateWorkSet*() fonksiyonu için sözde kod algoritma 1 ile verilmiştir.

LLFPC ve alt görev kümesi yaratma algoritmasının çalışmalarını kapsayan görevlerin durum geçiş şeması Şekil 3.4'te verilmiştir.

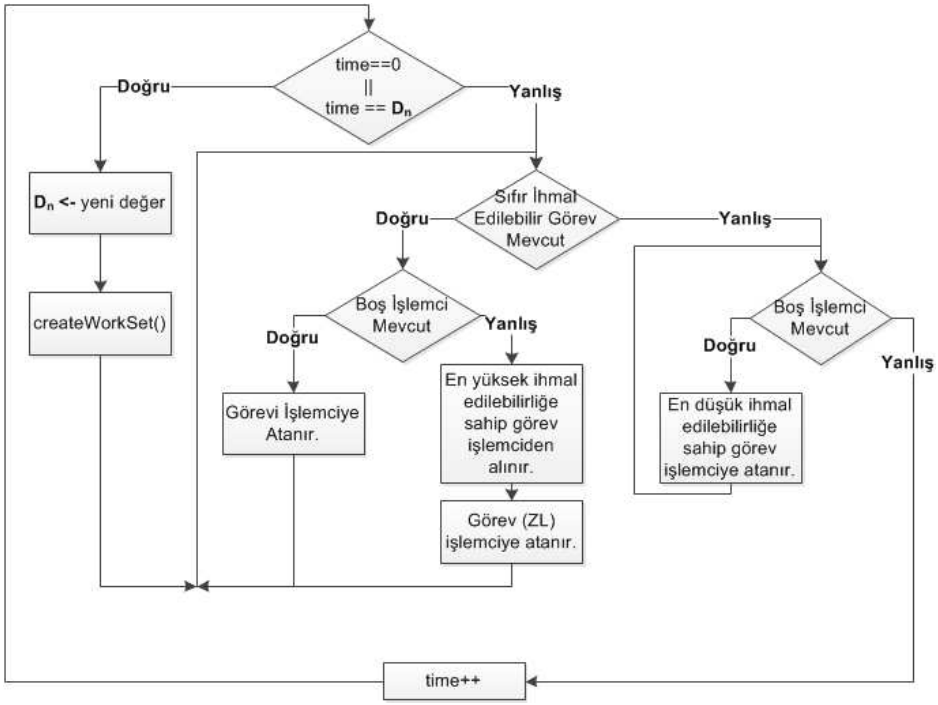
Algorithm 1 CreateWorkSet(GTS, D_n)

Calculate $totalTimeSlots$
Assign time slots to HP tasks
Calculate $idleTimeSlots$
if $idleTimeSlots > 0$ **then**
  foreach(Task T in LP)
  if $laxity < D_n - t$ **then**
    Assign ($D_n - t - laxity$) time slot to T
    Update $idleTimeSlots$
  end if
end if
if $idleTimeSlots > 0$ **then**
  Distribute $idleTimeSlots$ to LP tasks by real laxities
end if



Şekil 3.4: VLDS algoritmasında işlerin durum geçiş şeması

Şekil 3.4'te de görüldüğü gibi yeni gelen her görev global görev listesine girmektedir. Her alt görev kümesi yaratma işlemi çalıştırıldığında bu kümeden alt bir küme güncel görev kümesi olarak atanmakta ve kalan görevler kalan görev listesinde tutulmaktadır. Global görev listesinde görevler zaman sınırlarına göre sıralı tutulmaktadır. Güncel görev kümesindeki tüm durum değişimleri LLFPC algoritması tarafından yapılmakta ve buradaki tüm listeler tamamen sanal değerlere dayalı çalışmaktadır. Örneğin sanal çalışma süresini tamamlayan bir görev gerçekte halen çalışma süresine ihtiyaç duyuyor olsa bile işlemciden alınmaktadır. Güncel çalışma kümesinin yaratıldığı her aralık sonuna gelindiğinde gerçek kalan çalışma sürelerine sahip görevler tekrar global görev listesine eklenmektedirler.



Şekil 3.5: VLDS akış diyagramı

İş sıralama algoritmasının her kuantumda verdiği kararları gösteren akış diyagramı Şekil 3.5'te verilmiştir. Burada da görülebildiği gibi iş sıralama algoritması *createWorkSet()* metodunu yalnızca en yakın zaman sınırına ulaşıldığında (D_n) kullanmaktadır. Bunun dışındaki durumlarda boş işlemci bulunuyorsa veya sıfır ihmal edilebilirlik değerine sahip iş bulunuyorsa iş sıralama algoritması işlem yapmaktadır. Bu durumların hiç birinin olmadığı anlarda iş sıralama algoritması tarafından bir işlem gerçekleştirilmesi gerekmemektedir.

3.3 Doğruluk Tanıtı

Yüksek Öncelikli (HP) görevlere gerekli kaynak ataması yapıldıktan sonra kalan işlemci kaynağı Düşük Öncelikli (LP) görevler için gerekli olan kaynak sayısından düşük ise işlerden bazıları tanımlanan aralıkta negatif ihmal edilebilirlik değerine sahip olacak ve zaman sınırlarını kaçıracaktırlar.

Adım 1: Tanıtta bu varsayımdan (totalTimeSlotsRequiredforLPTask $\geq$ idleTime) yola çıkılarak $t = 0$ anında bu durumun oluşabileceği görev kümelerinde toplam görev yükünün sistemde bulunan işlemci sayısından büyük olduğu gösterilmiş. Dolayısıyla

bu görev kümelerinin zaman sınırını kaçırdığı sistemdelerde bulunan
işlemci sayısı ile başarılı iş sıralaması yapılmasının mümkün olmadığı tanıtlanmıştır.

$T_j^{HP} \in HP \wedge T_k^{LP} \in LP$ varsayımı ile, zaman kısıtlarının algoritma tarafından sağlanamadığı koşul aşağıdaki denklemde verildiği gibi yazılabilir.

$$\sum_{k=0}^{|LP|} \left[D_n - l_{T_k^{LP}} \right] \geq m \cdot D_n - \sum_{j=0}^{|HP|} e_{T_j^{HP}} \, . \tag{3.9}$$

$$\sum_{k=0}^{|LP|} \left[\frac{D_n - d_{T_k^{LP}} + e_{T_k^{LP}}}{D_n} \right] \geq \frac{m \cdot D_n - \sum_{j=0}^{|HP|} e_{T_j^{HP}}}{D_n} \, . \tag{3.10}$$

$$\sum_{k=0}^{|LP|} \left[1 + \frac{e_{T_k^{LP}} - d_{T_k^{LP}}}{D_n} \right] \geq m - \sum_{j=0}^{|HP|} \frac{e_{T_j^{HP}}}{D_n} \, . \tag{3.11}$$

$$\sum_{k=0}^{|LP|} \left[1 + \frac{\left(u_{T_k^{LP}} - 1 \right) \cdot d_{T_k^{LP}}}{D_n} \right] + \sum_{j=0}^{|HP|} u_{T_j^{HP}} \geq m \, . \tag{3.12}$$

Düşük Öncelikli görev kümesinin tanımından (Denklem **3.4**), aşağıdaki denklem yazılabilir.

$$d_{T_k^{LP}} > D_n \Rightarrow \frac{d_{T_k^{LP}}}{D_n} = 1 + \varepsilon \;\; for \;\; \varepsilon > 0 \, . \tag{3.13}$$

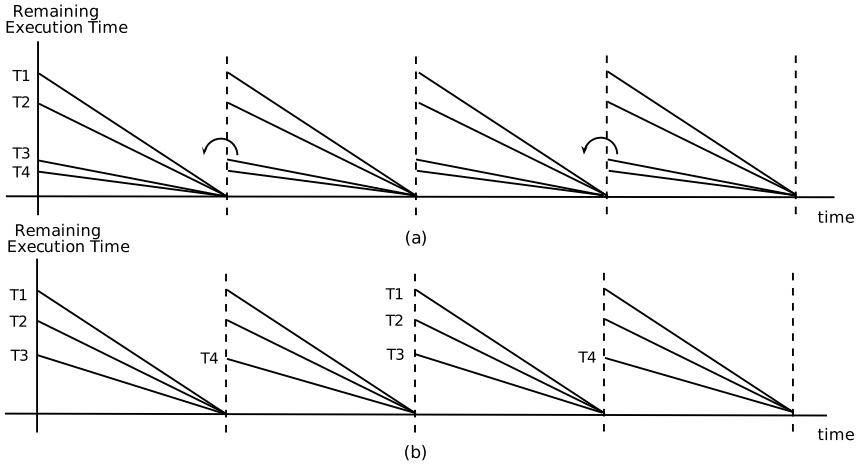
$$\sum_{k=0}^{|LP|} \left[1 + \left(u_{T_k^{LP}} - 1 \right) (1 + \varepsilon) \right] = \sum_{k=0}^{|LP|} \left[u_{T_k^{LP}} + \varepsilon \cdot \left(u_{T_k^{LP}} - 1 \right) \right] \, . \tag{3.14}$$

$$\sum_{k=0}^{|LP|} \left[u_{T_k^{LP}} + \varepsilon \cdot \left(u_{T_k^{LP}} - 1 \right) \right] < \sum_{k=0}^{|LP|} u_{T_k^{LP}} \, . \tag{3.15}$$

$$\sum_{k=0}^{|LP|} u_{T_j^{LP}} + \sum_{j=0}^{|HP|} u_{T_j^{HP}} > m \, . \tag{3.16}$$

$$\sum_{i=0}^{|GTS|} u_{T_i} > m \, . \tag{3.17}$$

Denklem **3.17** ile gösterildiği gibi zaman sınırlarının kaçırıldığı görev kümelerinin, toplam yükleri sistemde bulunan işlemci sayısından yüksektir. Dolayısıyla zaman sınırları kaçırılmadan iş sıralaması yapılması mümkün değildir.



Şekil 3.6: Akıcı iş sıralamada çalışma süreleri dağıtımı (a: Adil dağıtım, b: Yer değiştirme sonrası)

Adım 2: İkinci adımda, ispat oluşturulan diğer alt görev kümeleri için genişletilmiştir. Akıcı iş sıralama yönteminde görevlerin çalışma sürelerinin, aralıklara adil şekilde dağıtımı ile başarılı bir gerçek zamanlı iş sıralaması oluşturmak mümkündür. Şekil 3.6'de görüldüğü gibi önerilen algoritma ihmal edilebilirlik değerlerini de göz önünde bulundurarak alt görev kümesinin oluşturulduğu mevcut aralık içersinde zaman sınırı uzak olan periyodik görevler arasında iş yükü değişimi yapmaktadır. Böylece adil bir dağıtımdan uzaklaşılmakta fakat işlerin ortalama tepki süreleri arttırılmaktadır.

4. GERÇEKLEŞTİRİM

Sanal İhmaledilebilirlik Güdümlü İş Sıralaması (VLDS) algoritmasının gerçeklenmesinde STORM (Simulation Tool for Real Time Multiprocessor scheduling) benzetim programı kullanılmıştır [1]. Bu bölümde benzetim programının yapısı ve kullanımı anlatılmıştır.

4.1 Simulasyon Ortamı

STORM java geliştirme ortamı sürüm 1.6 kullanılarak geliştirilmiştir. Bu sayede farklı işletim sistemleri üzerinde kullanılabilmektedir. STORM gerçek zamanlı iş sıralama algoritmaları için ayrık zamanlı bir benzetim ortamı ve raporlama arayüzü sunmaktadır. Benzetim ortamında uygulacak senaryolar programa XML dosyaları ile aktarılmakta ve sonuçlar kullanıcı arayüzü veya oluşturulan çıktı dosyalarıyla incelenebilmektedir.

4.1.1 Görev ve kaynak modelleri

STORM benzetim programında iş sıralayıcı çoğunlukla görev ve işlemci modeli olmak üzere iki temel model ile ilişki içerisindedir.

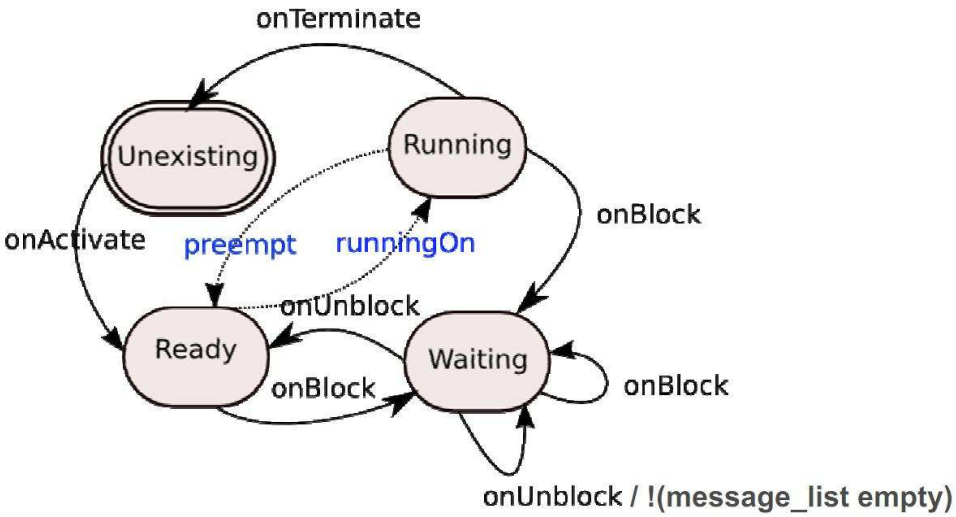
Görev modelinden yaratılan nesneler bilgilerini benzetim başlatıldığında XML dosyasından almaktadırlar. Benzetim programında görevlerin yalnızca belirtilen süre kadar kaynak kullandıkları düşünülmekte ve benzetim programı bu bilgilere dayanarak yürütülmektedir. Görevlerin tanımladıkları, bu süre bilgileri dışında içerdikleri fonksiyonel özellikleri bulunmamaktadır. Yani görevler içerisinde herhangi bir dizgi(kod) yürütülmemektedir. Benzetim programının girdi olarak aldığı örnek bir xml dosyası Dizgi 4.1'de verilmiştir. Burada da görüldüğü gibi xml dosyası iki temel kısımdan oluşmaktadır. Bunlardan ilki işlemci grubunu tanımlamakta, ikincisi ise görev grubunu tanımlamaktadır. Görevlere ait en yüksek çalışma süreleri ve periyot değerleri burada *WCET*, *period* özellikleri (*attribute*) olarak belirtilmiştir.

Dizgi 4.1: Örnek XML dosyası

```
<SIMULATION duration="12">

  <SCHED className="storm.Schedulers.Pfair" quantum="1"></SCHED>
  <CPUS>
    <CPU className="storm.Processors.CT11MPCore"
        name ="CPU C" id="2"  ></CPU>
    <CPU className="storm.Processors.CT11MPCore"
        name ="CPU B" id="1"  ></CPU>
    <CPU className="storm.Processors.CT11MPCore"
        name ="CPU A" id="0"  ></CPU>

  </CPUS>

  <TASKS>
    <TASK className="storm.Tasks.PTask_NAM" name ="T_a1" period="2"
      id="1" WCET="1" count="0" url="./TASKT1.jpg" priority="1"/>
    <TASK className="storm.Tasks.PTask_NAM" name ="T_a2" period="2"
      id="2" WCET="1" count="0" url="./TASKT2.jpg" priority="1"/>
    <TASK className="storm.Tasks.PTask_NAM" name ="T_b1" period="3"
      id="3" WCET="2" count="0" url="./TASKT3.jpg" priority="1"/>
    <TASK className="storm.Tasks.PTask_NAM" name ="T_b2" period="3"
      id="4" WCET="2" count="0" url="./TASKT4.jpg" priority="1"/>
    <TASK className="storm.Tasks.PTask_NAM" name ="T_b3" period="3"
      id="5" WCET="2" count="0" url="./TASKT5.jpg" priority="1"/>
  </TASKS>


</SIMULATION>
```

Benzetim programı içerisinde görevler için durum geçiş diyagramı Şekil 4.1'te verilmiştir. Bir görev benzetim başlangıcında *Unexisting* durumunda bulunmakta ve ilk isteği geldiğinde *Ready* durumuna geçmektedir. İşlemciye sahip olduğunda ise *Running* durumunda bulunmaktadır. Periyodik bir görevin bir periyot içerindeki işi (job) tamamlandığında bu görev *Waiting* durumuna geçmektedir. Bu geçiş sırasında Şekil 4.1'de de görüldüğü iş sıralayıcı tarafından kullanılacak belirli olaylar tetiklenmektedir. Bu çalışmada önerilen algoritma yaklaşım olarak durum geçiş diyagramı içerisindeki *Ready* durumunu alt durumlara bölmüş ve işlemci kullanımına bu alt durumlar arasından yaptığı seçimler ile karar vermiştir. Bu alt durumların gerçeklenmesiyle ilgili ilerleyen kısımlarda detaylı bilgi verilecektir.

STORM benzetim programında çok detaylı bir işlemci modeli bulunamamaktadır. Benzetim programı tarafından işlemciler arasındaki ilişkiler veya cep bellek mimarisi



Şekil 4.1: STORM benzetim programında görevlerin durum geçiş diyagramı [1]

modellenmemiştir. Benzetim sürecinde gerçeklenecek senaryonun tanımlandığı XML dosyasında işlemci ile ilgili yalnızca bazı temel bilgiler tanımlanmaktadır.

Simulasyon gerçek zamanlı çalışan bir ortamı sunmamakta, belirnen bir zaman aralığı için işlerin hangi işlemcileri kullanacağını hesaplayarak, çıktı olarak vermektedir.

4.1.2 İş sıralayıcı modeli

STORM programına girdi olan verilen xml dosyası içerisinde görevler ve işlemciler ile ilgili bilgilerin yanı sıra simulatorun hangi iş sıralama algoritmasını kullanacağına dair bilgiyi de içermektedir. Böylece simulasyon programına, oluşturulan iş sıralama algoritmasına içeren sınıf ile tekrar derlenerek, yeni algoritmalar eklenebilmekte ve xml içerisinde bu iş sıralayıcılara ait algoritmalar adreslenerek kullanılabilmektedir.

İş sıralayıcı temel sınıfının sahip olduğu ve iş sıralayıcının düzgün çalışabilmesi için tekrar yazılması gereken metotların listesi ve bu methodlara ilişki bilgiler kısaca aşağıda verilmiştir.

- *init()*: Bu metotta algoritma içerisinde kullanılacak olan listelerin tanımlanması gibi iş sıralayıcıya ait başlangıç işlemleri gerçekleştirilmektedir.
- *onTick()*: Bu metot her kuantum sonrasına simulator tarafından çağrılmaktadır. İşlemciden yer alan işlerin kalan çalışma sürelerinin düşürülmesi gibi her adım sonrasında yapılması gereken işlemleri içermektedir.

- *sched()*: Her kuantumun başlangıcında çağrılan bu metotta iş sıralama algoritmasının o kuantum içersinde çalışıp çalışmayacağına karar verilmekte ve işlerin tutulduğu listelerde güncellemeler yapılmaktadır.

- *select()*: Bu metotta iş sıralamada yapılacak olan tercih işlemler ve bu işlemleri içeren diğer metotlara çağrılar yapılmaktadır. *select()* metotuna çağrı *sched()* metotundan yapılmakta, bu metot her kuantumda çalışmamaktadır.

- *onActivate()*: Bir göreve ait ilk iş başladığında ($t = 0$ anında), o işi parametre olarak alan bu metot simulator tarafından çağrılmaktadır. Metot içerisinde, aktif hale getirilen iş ilgili listeye alınarak, iş sıralama algoritmasında yer alması sağlanmaktadır.

- *onBlock()*: Bir göreve ait işin çalışma süresini tamamlaması sonucunda bu metot simulator tarafından çağrılmaktadır. Bu metot içerisinde sonlanan işe ait bilgiler iş sıralama algoritmasının kullandığı listelerden çıkartılmaktadır.

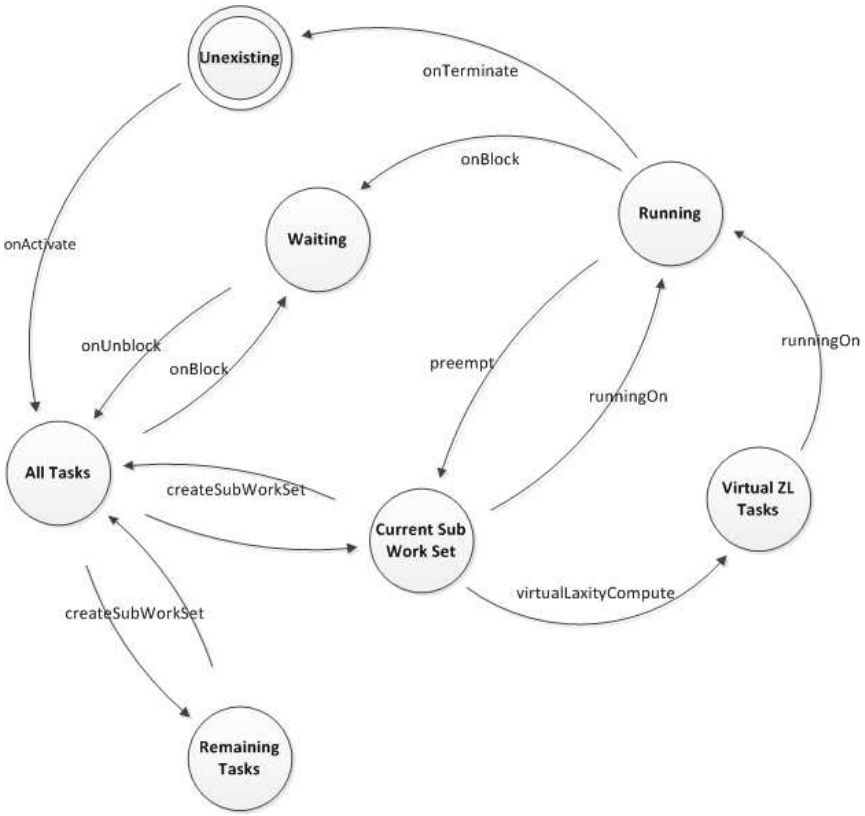
- *onUnBlock()*: Bir göreve ait yeni bir iş geldiğinde bu metot simulator tarafından çağrılmaktadır.Metot içerisinde, aktif hale getirilen iş ilgili listeye alınarak, iş sıralama algoritmasında yer alması sağlanmaktadır.

- *onTerminate()*: Bir göreve ait tüm işler sonlandığında bu metot çağrılmaktadır. Periyodik görevlerin iş sayısında bir sınır bulunmadığından bu metota çağrı yalnızca simulasyon sonlanırken yapılmaktadır.

4.2 Önerilen Algoritmanın Gerçekleştirimi

Simulasyona eklenen iş sıralama algoritmalarının içermesi gereken ve simulator tarafından kullanılan metotlar dışında geliştirilen iş sıralama algoritması bazı *private* metotlar içermektedir.

- *CreateWorkSet*: Metotu en yakın zaman sınırına göre alt görev kümeleri oluşturulmak için iş sıralayıcı tarafından kullanılmaktadır. İş sıralayacı $t = 0$ anında ve zaman sınırlarına gelindiğinde bu metota çağrı yapmaktadır.



Şekil 4.2: VLDS algoritmasında görevlerin durum geçiş diyagramı

- *LaxityCompute*: İhmal edilebilirlik (laxity) değeri ve diğer sanal değerler simulasyonun görev modeli içerisinde bulunmayan değişkenlerdir. Bu değişkenler görev modeli içerisinde bulunan hash tablosunda tutulmaktadır. Bu yüzden bu değerler kalan çalışma süresi, bağıl zaman sınırı gibi simulator tarafından güncellenememektedirler. Bu metot ihmal edilebilirlik değerine göre çalışan listelerdeki görevlerin ihmal edilebilirlik değerlerini her kuantumda güncellemektedir.

- *VirtualLaxityCompute*: Bu metot da bir önceki metot gibi hash tablosunda bulunan sanal ihmal edilebilirlik değerini, bu değeri kullanan listelerde bulunan görevler için güncellemektedir. Bu değer de simulator tarafından güncellenmediğinde dolayı bu metodunda her kuantum çalıştırılarak listelerdeki sıraların ve bilgilerin güncel tutulması sağlanmaktadır.

4.2.1 Görev listeleri

Şekil 4.2'de VLDS algoritmasının sahip olduğu durum geçiş şeması verilmiştir. Görüldüğü gibi *Ready* listesi alt listelere parçalanmış ve işlemciye sahip olacak işler

bu listelere arasından seçilmeye başlanmıştır.Burada *Ready* listesinin yerine gelen listelerin özellikleri ve işlevleri ile ilgili kısaca bilgiler aşağıda verilmektedir.

- *All Tasks*: Algoritmanın tanımında GTS olarak adlandırılan listenin yazılımda temsilidir. Sistemde bulunan tüm işleri başlangıç durumunda içeren listedir. *CreateWorkSet* metotu bu liste içerisinde bulunan tüm işler diğer listelere taşınır. Alt iş kümesinin tanımlandığı aralık sona erdiğinde tüm görevler tekrar bu listeye alınır. Bu liste içerisinde bulunan işler zaman sınırlarına göre sıralanmaktadır. Böylece bu liste içerisindeki ilk elemana erişim yapılarak en yakın zaman sınırı bulunabilmektedir.

- *Current Sub Work Set*: Tanımlanan aralık içerisinde işlemciye sahip olacak işleri içeren kümedir. Bu küme içerisindeki görevler işlemciye doğrudan sahip olabilecekleri gibi *Virtual ZL Tasks* listesine girerek oradan da işlemciye sahip olabilirler. Bu küme içerisindeki görevler sanal ihmal edilebilirlik değerlerine göre sıralanmakta ve en düşük sanal ihmal edilebilirlik değerine sahip işler *Running* durumuna geçmekte yani işlemciye sahip olmaktadır.

- *Remaining Tasks*: Bu liste en yakın zaman sınırıyla belirlenen zaman aralığında işlemciye sahip olmayacak işleri içermektedir. Bu aralık sona erdiğinde bu işlemler tekrar *All Tasks* listesine katılarak iş sıralamaya dahil olmaktadırlar.

- *Virtual ZL Tasks*: LLFPC algoritmasının çalışması sırasında sanal ihmal edilebilirlik değerleri sıfır olan görevleri içermektedir. Burada bulunan görevler için işlemci üzerinde (*Running* durumunda) olan en yüksek ihmal edilebilirlik değerine sahip görev işlemciden alınmakta (preempt) ve yerine bu listede yer alan iş, işlemciye atanmaktadır.

4.3 Önerilen Algoritmanın Simulasyona Eklenmesi

Benzetim programı içerisinde bulunan iş sıralama sınıfından türetilerek yazılan yeni sınıf dosyası benzetim programı (storm.jar) ile birlikte derlenmekte ve program içersinde bulunan iş sıralayıcıları yoluna eklenmektedir. Bu işlemleri gerçekleştirmesi için yazılmış olan ve eklenecek sınıfın kaynak kodunu parametre olarak alan komut dosyası (*script*) Dizgi 4.2’de verilmiştir.

Dizgi 4.2: Komut Dosyası

```
# $1 : Scheduler java source code file

echo "Compiling ..."
mkdir .temp
javac -cp storm-3-2.jar $1 -d ./.temp

echo "Updating jar file ..."
cd ./.temp
jar uvf ../storm-3-2.jar ./storm/Schedulers/*.class

echo "Removing Temporary Files ..."
cd ..
rm -R .temp
rm LogFile.log

echo "Running Simulation ..."
java -cp storm-3-2.jar programme.programme

exit
```

Komut dosyası çalıştırıldıktan sonra benzetim programı içersinde bulunan terminal ile girdi olarak alınacak xml dosyasının adresi verilmektedir. Xml dosyası içerisinde bulunan ikinci satırda benzetim programı içerisinde bulunan hangi iş sıralayıcının kullanıcağı belirtilmektedir. Komut dosyasının yürütülmesi ile eklenmiş olan sınıf, girdi olarak verilen xml tarafından seçilerek benzetim programının geliştirilen algoritma ile çalıştırılması sağlanmaktadır.

5. DENEYSEL SONUÇLAR

Bu bölümde STORM simulasyon ortamında geliştirilen algoritmalar kullanılarak tasarlanan test senaryoları üzerinde bu algoritmalarun ürettikleri sonuçlar karşılaştırılmıştır.

5.1 Gerçeklenen Algoritmalar

STORM simulasyon ortamında VLDS algoritması dışında Pfair [9] ve Boundary Fair [13] algoritmaları gerçeklenmiş ve üretilen test senaryolarında bu algoritmaların ürettikleri sonuçlar karşılaştırılmıştır. STORM ayrık zamanlı bir çalışma ortamı sunduğundan ve TL plane ve türevlerinin algoritmasının %100 yüklü sistemler için Bfair ile aynı sonucu üretmesinden dolayı bu algoritmalar simulasyon ortamında gerçeklenmesine gerek duyulmamıştır.

5.2 Test Senaryoları

VLDS yalnızca %100 yüklü test senaryoları için diğer algoritmalar ile karşılaştırılmıştır. İlk senaryoda amaç en kötü durumda bile VLDS algoritmasının iş sıralaması işlemini başarıyla gerçekleştirdiğini göstermektir. İkinci senaryoda ise algoritmanın çalışmasına daha uygun bir test kümesi seçilmiş ve algoritmanın sağladığı performans artışı literatürdeki diğer algoritmalarla karşılaştırılmıştır.

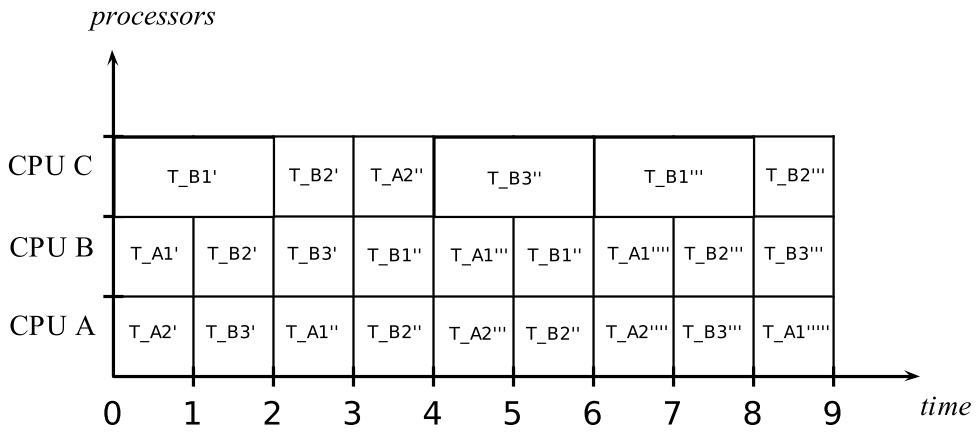
5.2.1 En kötü durum senaryoları

Bu senaryoda görev kümesinin başarılı şekilde iş sıralamasının yapılabilmesi için görevler arasında adil bir kaynak paylaşımının gerekli olması koşulu amaçlanmıştır. VLDS adil kullanım (fairness) kısıtlarına sahip olmamasına karşın bu gibi görev setlerinde de adil dağıtım sağlayarak başarılı iş sıralamasını gerçekleyebilmektedir.

Bu test senaryosunun görev kümesi belirlenirken dikkat edilen unsur en aralıkta düşük öncelikli (LP) ve yüksek öncelikli (HP) görev kümeleri için yapılması gereken iş

Görev Adı	Çalışma Süresi	Periyot (Zaman Sınırı)
T_A1	1	2
T_A2	1	2
T_B1	2	3
T_B2	2	3
T_B3	2	3

Çizelge 5.1: En kötü durum senaryosu için görev kümesi



Şekil 5.1: En kötü durum senaryosuna ait görev kümesi için oluşturulan iş sıralama planı

yükü atamalarının en yüksek seviyede olmasıdır. Yapılması zorunlu bu atamalar gerçekleştirildikten sonra kalan boş işlemci süresi 1'e eşittir.

Daha önceki kısımlarda da bahsedildiği gibi VLDS algoritmasının sahip olduğu avantajlar yapılması zorunlu atamalar gerçekleştirildikten sonra geriye kalan boş işlemci süresinin dağıtımı ile sağlanmaktaydı. Bu ve benzeri senaryolarda bu süre en alt seviyeye çekildiğinden algoritma muadillerine göre bir avantaj sağlayamamakta ama tüm görevleri zaman sınırları içerisinden tamamlamayı başarmaktadır. Bu testteki amaç algoritmanın en kötü durumunda bile muadilleriyle aynı iş planını üretebildiğini göstermektir.

Bu senaryoda VLDS algoritması tarafında kullanılacak olan görev kümesi çizelge 5.1'te verilmiştir.

Çizelge 5.1'te verilen görev kümesi için VLDS algoritmasının ürettiği iş planı şekil 5.1'te verilmiştir. Görüldüğü gibi adil davranan (fair) algoritmalar ile (Pfair, Bfair) aynı sonucu üretmiştir.

Görev Adı	Çalışma Süresi	Periyot (Zaman Sınırı)
T_A1	3	5
T_A2	3	5
T_B1	4	20
T_B2	4	20
T_B3	4	20
T_B4	4	20

Çizelge 5.2: İyileştirme sağlanan durum senaryosu için görev kümesi

5.2.2 Daha iyi iş sıralaması sağlanan senaryolar

Bu senaryolarda seçilen görev kümesinde alt görev kümelerinin yaratıldıkları aralıklarda zorunlu atamalar yapıldıktan sonra kalan boş işlemci sürelerinin yüksek olması hedeflenmiştir. Örnek olarak verilen görev kümesinde zorunlu atamalar yalnızca HP görevler için gerekmekte LP görevler için işlemci paylaşımı algoritma tarafından herhangi bir adil davranma kısıtına bağlı olmaksızın gerçeklenmektedir.

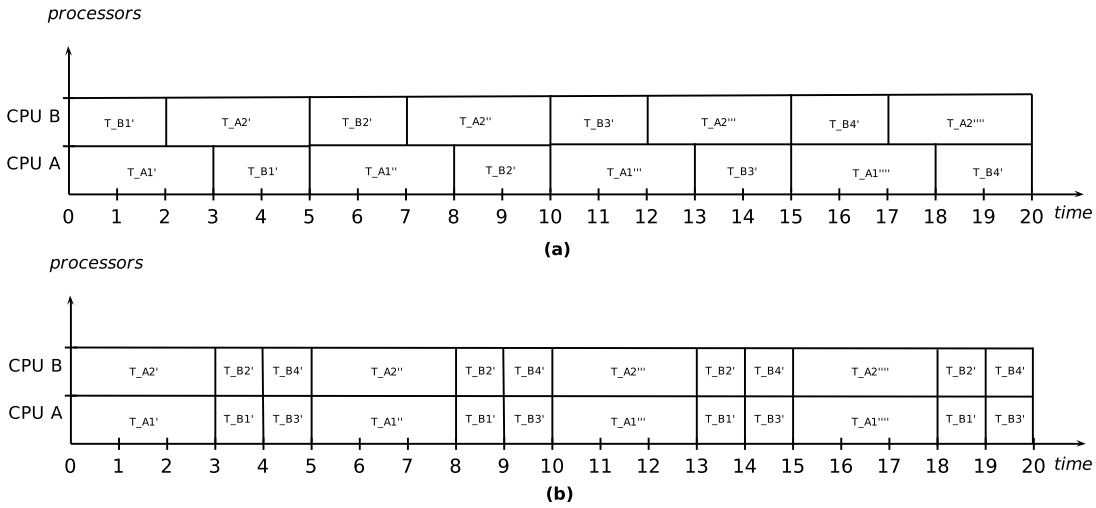
Bu senaryoda kullanılan görev kümesi çizelge 5.2'te verilmiştir.

Çizelge 5.2'te verilen görev kümesi için VLDS algoritmasının ve BFair algoritmasının ürettiği iş planı şekil 5.2'te verilmiştir. İncelenen senaryoda görüldüğü gibi görevlerin çalışmalarının parçalanması azalmıştır ve bir super period içerisnde bağlam değişim sayısı 32'den 8'e düşmüştür.

Burada her zaman sınırı içerinde adil davrama (fairness) kısıtları olan BFair tüm görevlere bu aralıklar içerisinde oransal şekilde yer vermiştir. Bu yüzden zaman sınırı 20 olan tüm görevler 15 ile 20 arasında sonlanmıştır. VLDS algoritmasında ise görevlerin parçalanması azaltılarak zaman sınırı 20 olan görevlerin ortalama olarak daha erken sonlanması sağlanmıştır.

5.3 Sonuçlar

İki senaryo için algoritmanın çalışması diğer algoritmalar ile karşılaştırılmıştır. Birinci senaryoda tüm görevleri zaman sınırları içerisinde tamamlayabilmek için tek olası plan tüm işlerin adil şekilde işlemcilere sahip olabilmesi ile mümkündür. VLDS algoritması



Şekil 5.2: İyileştirilmiş senaryoya ait görev kümesi için oluşturulan iş sıralama planı (a: VLDS algoritması, b: BF algoritması)

en kötü durumun senaryoları için adil davranma kısıtlarına sahip olan algoritmalarla aynı planı üretebilmektedir.

İkinci senaryoda ise adil davranma kısıtlarının esnetebileceği görev kümelerine örnek verilmiştir. Bu görev kümesinde görev değerleri belirlenirken HP ve LP işlere zorunlu yapılması gereken atamaların düşük seviyede tutulması amaçlanmıştır. Zorunlu atamalar sonucunda boş kalan sürenin diğer işler arasında parçalanmayı azaltacak şekilde dağıtımı aynı iş sıralama algoritması sınıfına ait literatürdeki diğer algoritmalardan daha iyi düşük masraf ile iş sıralaması yapmasın olanak vermektedir.

6. SONUÇ VE ÖNERİLER

Gerçek zamanlı çok işlemcili sistemlerde örtülü zaman sınırlı (implicit deadline) periyodik görevler için adil davranma (fairness) kısıtları bulundurmayan bir iş sıralama algoritması sunulmuştur. VLDS algoritması literatürdeki benzerlerine göre %100 yüklü (utilized) görev kümelerinde daha iyi iş sıralamaları oluşturabilmektedir. Burada daha iyi iş sıralamaları ile belirtilmek istenilen kavram daha az bağlam değişimi yapılması ve görevlerin ortalama tepki sürelerinin düşürülmesidir.

VLDS algoritmasının bundan sonraki çalışmalarda ULTRASPARC IV işlemcisi üzerinde yardımcı işlemci (co-processor) olarak gerçeklenmesi ve çalışmanın teorikten pratiğe geçilirmesi planlanmaktadır.

Algoritma simulasyon ortamında geliştirilip test edildiğinden bazı maliyetler göz ardı edilmiştir. Simulasyon ortamında cep bellek ve ilgili işlemler algoritmaya dahil edilmemiş ve işlemci değiştirme maliyeti göz ardı edilmiştir. VLDS algoritmasında alt iş kümeler içerisinde iş sıralama yaparken ve bu alt iş kümeler arasında geçiş işlemleri sırasında işlemcilerin sahip oldukları cep bellek içerikleri göz önünde bulundurularak atama yapılabilir. Böylece sistemin cep bellekte meydana gelecek ıskalar sonucun tepki süresinin uzaması önlenmiş olacaktır. Bunun dışında işlemci değiştirmeden (migration) yapılabilecek olası çözümler olduğu durumda bağlamın başka bir çekirdeğe veya işlemciye taşınması maliyetinden kaçınmak için algoritmada eklentiler yapılabilir. Bu şekilde simulasyon ortamında göz ardı edilen fakat algoritmanın pratikte kullanımda meydana gelecek maliyet azaltılmış olacaktır.

KAYNAKLAR

[1] **Urunuela, R.**, **Deplanche, A.M. ve Trinquet, Y.**, (2009). Simulation TOol for Real-time Multiprocessor scheduling.

[2] **Burns, A. ve Wellings, A.J.**, (2001). Real-Time Systems and Programming Languages: ADA 95, Real-Time Java, and Real-Time POSIX, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd sürüm.

[3] **Pinedo, M.L.**, (2008). Scheduling: Theory, Algorithms, and Systems, Springer Publishing Company, Incorporated, 3rd sürüm.

[4] **Liu, C.L. ve Layland, J.W.**, (1973). Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *J. ACM*, **20(1)**, 46–61.

[5] **Carpenter, J.**, **Funk, S.**, **Holman, P.**, **Srinivasan, A.**, **Anderson, J. ve Baruah, S.**, (2004). A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms, Handbook on Scheduling Algorithms, Methods, and Models, Chapman Hall/CRC, Boca.

[6] **Leung, J.**, (1989). A new algorithm for scheduling periodic, real-time tasks, *Algorithmica*, **4**, 209–219, 10.1007/BF01553887.

[7] **Davis, R.I. ve Burns, A.**, A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems.

[8] **Andersson, B. ve Tovar, E.**, (2006). Multiprocessor Scheduling with Few Preemptions, IPP Hurray Research Group, Polytechnic Institute of Porto, Portugal HURRAY-TR-060811.

[9] **Baruah, S.K.**, **Cohen, N.K.**, **Plaxton, C.G. ve Varvel, D.A.**, (1996). Proportionate progress: A notion of fairness in resource allocation, *Algorithmica*, **15**, 600–625.

[10] **Dertouzos, M. ve Mok, A.**, (1989). Multiprocessor online scheduling of hard-real-time tasks, *Software Engineering, IEEE Transactions on*, **15(12)**, 1497 –1506.

[11] **Baruah, S.K.**, **Gehrke, J.E. ve Plaxton, C.G.** Fast Scheduling of Periodic Tasks on Multiple Resources, In Proceedings of the 9th International Parallel Processing Symposium, s.280–288.

[12] **Anderson, J.H. ve Block, A.**, (2000). Early-release fair scheduling, In Proceedings of the 12th Euromicro Conference on Real-Time Systems, s.35–43.

[13] **Zhu, D.**, **Mosse, D. ve Melhem, R.**, (2003), Multiple-Resource Periodic Scheduling Problem: How Much Fairness is Necessary?

[14] **Cho, H.**, **Ravindran, B. ve Jensen, E.D.**, (2006). An Optimal Real-Time Scheduling Algorithm for Multiprocessors, RTSS, s.101–110.

[15] **Funaoka, K.**, **Kato, S. ve Yamasaki, N.**, (2008). Work-Conserving Optimal Real-Time Scheduling on Multiprocessors, s.13–22.

EKLER

EK A : Terimler Sözlüğü

EK A

Çizelge A.1: Terimler Sözlüğü

İngilizce	Türkçe
Scheduling	İş sıralama (Çizelgeleme)
Laxity	İhmal edilebilirlik
Task	Görev
Job	İş
Deadline	Zaman sınırı
Fairness	Adil davranma
Rate Monotonic	Oransal Monoton
Earliest Deadline First	En Yakın Zaman Sınırı Önce
Least Laxity First	En Az İhmal Edilebilir Önce
Preemption	Kesilme
Migration	İşlemci Değiştirme
Fluid Scheduling	Akıcı çizelgeleme
Work Conserving	İş Odaklı

ÖZGEÇMİŞ

Ad Soyad: Gökhan Seçinti

Doğum Yeri ve Tarihi: Gaziantep, 19 Ağustos 1987

Adres: Şişli - İstanbul

E-Posta: secinti@itu.edu.tr

Lisans: ITU Bilgisayar Mühendisliği 2009

TEZDEN TÜRETİLEN YAYINLAR/SUNUMLAR

- **Seçinti G.**, Altılar D.T.: Virtual Laxity Driven Scheduling *The 5th International Conference on Communications, Computers and Applications*, 12-14 October 2012, Istanbul, Turkey (Submitted)