Bilgisayar Bilimleri ve Mühendisliği Dergisi

Cilt-18
Sayı-2
2025

PusOS: RISC-V Tabanlı Açık Kaynak Hafif Bir Sis Bilişim Sistemi Tasarımı

PusOS: Building an Open Source Lightweight Fog Computing System on RISC-V

Muhammed Numan İNCE
Akdeniz Üniversitesi
Bilgisayar Mühendisliği
Antalya, Türkiye
numan.ince@alanya.edu.tr
ORCID: 0000-0003-0256-3010

Melih GÜNAY
Akdeniz Üniversitesi
Bilgisayar Mühendisliği
Antalya, Türkiye
mgunay@akdeniz.edu.tr
ORCID: 0000-0001-5409-6720

Öz

Sis bilişim, veriye yakın düğümlerde işlem yaparak düşük gecikme ve yüksek hizmet kalitesi sunmayı hedefleyen dağıtık bir bilişim modelidir. Bu çalışmada, yalnızca çekirdek modunda çalışan, RISC-V mimarisi üzerinde geliştirilen, nanokernel tabanlı ve açık kaynaklı bir işletim sistemi sunulmaktadır. Sistem, kaynak kısıtlı cihazlarda düşük gecikme, yüksek güvenilirlik ve küçük sistem ayak iziyle çalışacak şekilde tasarlanmıştır. Nanokernel mimarisi sayesinde bağlam değiştirme ve sistem çağrısı yükleri ortadan kaldırılarak uygulamalar doğrudan çekirdek alanında çalıştırılarak verimlilik artırılmıştır. Ağ, depolama, hesaplama, sanallaştırma ve yönetim işlevleri entegre edilerek QEMU sanal makine öykünücüsünde test edilmiştir. Linux ile yapılan karşılaştırmalı deneylerde, önerdiğimiz sisteminin veri toplama, depolama ve çoklu görevde 4–5,5 kat daha hızlı olduğu gösterilmiştir. Böylece sis bilişim alanı için güvenilir, genişletilebilir ve hafif bir alternatif işletim sistemi sunulmaktadır.

Anahtar sözcükler: Sis bilişim, İşletim sistemleri, Uç bilişim, Sanallaştırma, RISC-V

Abstract

Fog computing is a distributed computing model that aims to provide low latency and high quality of service by processing data near the source nodes. In this context, we present a nanokernel-based, open-source fog computing operating system developed on the RISC-V architecture that runs solely in kernel mode. The system is designed to operate with low latency, high reliability and a minimal footprint on resource-constrained devices. By using the nanokernel architecture, context switching and system call overheads are eliminated and applications run directly in kernel space to enhance efficiency. Integrated functionalities for networking, storage, computation, virtualization, and management have been tested extensively using the QEMU emulator. Comparative experiments with Linux demonstrate that the system performs 4 to 5.5 times faster in serial data collection, storage, and multitasking thereby providing a reliable, extensible and lightweight alternative OS for fog computing domain.

Keywords: Fog computing, Operating systems, Edge computing, Virtualization, RISC-V

Makale Bilgileri	**Article Info**
Türü: Araştırma	Type: Research
Geliş tarihi: 28.07.2025	Received date: 28.07.2025
Kabul tarihi: 13.10.2025	Accepted date: 13.10.2025

Atıf/ to Cite (IEEE): M. N. İNCE, M. GÜNAY: PusOS: RISC-V Tabanlı Açık Kaynak Hafif Bir Sis Bilişim Sistemi Tasarımı : Bilgisayar Bilimleri ve Mühendisliği Dergisi, Cilt 18 2025 Sayı-2 1-12. Sf: 81-94, **DOI: 10.54525/bbmd.1753048**

1. Giriş

Teknolojideki hızlı ilerlemeler, internete bağlı çeşitli cihazların yalnızca veri tüketicileri değil, aynı zamanda veri üreticileri olarak da işlev görmesine zemin hazırlamıştır. Bu durum, yalnızca Nesnelerin İnterneti kapsamında değil, daha geniş bir bağlamda Bulut bilişim ve büyük veri kavramlarının yaygınlaşmasına öncülük etmiştir. Günümüzde veri depolama ve işleme işlemlerinin büyük çoğunluğu Bulut bilişim

altyapıları üzerinde gerçekleştirilmektedir. Bunun temel nedenleri arasında, sunucuların yüksek erişilebilirliği, kolay yapılandırılabilirliği, yönetimsel esneklik ve azalan donanım maliyetleri yer almaktadır [1].

Bununla birlikte, az sayıda Nesnelerin interneti cihazının yer aldığı ve düşük hesaplama gücü gerektiren uygulamalarda Bulut bilişim maliyet etkin ve pratik bir çözüm sunsa da artan trafik yoğunluğu ve hesaplama talepleri karşısında Bulut tabanlı servislerin performansı olumsuz etkilenmekte; maliyetler yükselmekte, ağ tıkanıklığı ve kaynak kısıtları nedeniyle gecikmeler artmaktadır. Özellikle coğrafi olarak dağıtılmış, binlerce cihazdan oluşan büyük ölçekli Nesnelerin İnterneti sistemlerinde 10 milisaniyenin altında yanıt süresi gerekliliği yalnızca Bulut çözümleriyle karşılanamayabilir [2]. Ek olarak, tüm verinin Bulut'a aktarılması gereksiz bant genişliği tüketimi yaratmakta ve güvenlik ile gizlilik risklerini beraberinde getirmektedir. Bu bağlamda, kaynakların ve hizmetlerin son kullanıcıya yakın konumlandırıldığı Uç (Edge) ve Sis (Fog) bilişim paradigmaları giderek önem kazanmaktadır [3]. Sis bilişim, Bulut bilişimin dezavantajlarını azaltmak üzere geliştirilmiş ara bir çözüm olarak görülmekte ve küçük ölçekli Bulut yapıları gibi ağ, veri depolama, hesaplama, yönetim, izleme ve sanallaştırma gibi temel hizmetleri sağlamaktadır [4].

Son yıllarda, gizlilik odaklı ve düşük gecikmeli veri işleme ihtiyaçlarının artmasıyla, bağlı cihazlara yakın, merkeziyetsiz sistemlerin önemi daha da belirginleşmiştir. Bu bağlamda, sis bilişim paradigması, yüksek hizmet kalitesi sunmayı hedefleyen önemli bir çözüm olarak öne çıkmaktadır [5].

Sis bilişim alanında geliştirilen mevcut platformların büyük çoğunluğu Linux tabanlı olup, uygulamalar kullanıcı alanında yürütülmektedir [6]. Örneğin, FogDirector ve Fora FCP gibi sistemler, genellikle kapalı kaynaklıdır ve entegrasyon için API sunmaktadır [7]. Her ne kadar bu platformlar kaynak yönetimi, görev zamanlama ve uygulama dağıtımı gibi işlevler sağlasa da kapalı kaynak yapıları ve kullanıcı alanı tabanlı çalışmaları, genişletilebilirlik ve performans açısından sınırlamalar getirmektedir [8].

Bu kısıtlamaların aşılması amacıyla sunucusuz bilişim ve çeşitli simülasyon/öykünme ortamları geliştirilmiştir. Sunucusuz bilişim, işlevlerin ağ kenarında çalıştırılmasına olanak tanısa da hala Bulut bağlantısına bağımlılık nedeniyle gecikmeler yaşanmakta ve doğrudan sis düğümlerinde yaygın kullanımı kısıtlıdır; ayrıca karmaşık yönetim süreçleri gerektirmektedir [9]. Simülasyon platformları ise hızlı prototipleme ve sistem doğrulama için kullanılsa da gerçek üretim ortamlarında çalıştırılmak üzere tasarlanmamıştır [10, 11].

Sonuç olarak, mevcut sis bilişim platformlarının büyük kısmı Linux tabanlı ve kullanıcı alanında çalıştığından, performans ve kaynak verimliliği açısından kısıtlamalarla karşılaşmaktadır.

Bu çalışmada, açık kaynaklı RISC-V komut seti mimarisi üzerinde geliştirilen, hafif ve özelleşmiş bir sis bilişim platformu olan Pus İşletim Sistemi (PusOS) tanıtılmaktadır. Çalışmanın temel katkıları aşağıdaki şekilde özetlenebilir:

- Sis bilişim ihtiyaçlarına özgü, gereksiz işlevsellikten arındırılmış özel amaçlı işletim sistemi tasarımı,
- Sistem çağrısı kaynaklı gecikmeleri ortadan kaldıran, kullanıcı alanına ihtiyaç duymadan doğrudan çekirdek alanında çalışan çalışma zamanı ortamı,
- RISC-V mimarisinin süpervizör kip avantajlarını kullanan, sanallaştırma desteğiyle güçlendirilmiş yalın çekirdek yapısı,
- Açık kaynaklı, kolay programlanabilir bir sistemin geliştirilmesi ve farklı projelerde test ve geliştirme platformu olarak kullanımı [12].

Çalışmanın geri kalanı şu şekilde düzenlenmiştir. Bölüm 2'de sis bilişimdeki güncel mimari yaklaşımlar incelenmekte; Bölüm 3, ilgili çalışmalar karşılaştırmalı olarak tartışılmakta; Bölüm 4'te PusOS'un tasarım ve mimari detayları sunulmakta; Bölüm 5 deneysel kurulum ve test senaryoları ile değerlendirmeyi içermekte; Bölüm 6'da sınırlılıklar ve güvenlik konuları tartışılmakta; son olarak, Bölüm 7'de çalışma sonuçları ve geleceğe yönelik araştırma önerileri ele alınmaktadır.

2. Sis Bilişimdeki Güncel Mimari Yaklaşımlar

Bu bölümde, sis bilişim platformları için en güncel işletim sistemi mimarileri ele alınmaktadır. Karşılaştırmalı değerlendirme, özellikle Linux tabanlı çözümler ile RISC-V mimarisi üzerine odaklanmaktadır.

2.1 Sis Bilişim İşletim Sistemi Olarak Linux

Bulut ile Nesnelerin İnterneti arasında yer alan Sis bilişim, bu ara katmanda görev yapan sistemlerin özgün ihtiyaçlarını karşılayabilecek esnek ve hafif işletim sistemlerine ihtiyaç duyar. Bu sistemlerin, yerel veri işleme ve düşük gecikmeli karar alma gibi işlemleri destekleyebilmesi beklenmektedir.

Nesnelerin İnterneti cihazları genellikle düşük maliyetli ve sınırlı donanım kaynaklarına sahiptir. Bu cihazlar çoğunlukla veri toplama ve temel kontrol işlevlerini üstlenirken, daha karmaşık analiz ve karar alma işlemleri genellikle merkezi Bulut altyapılarında gerçekleştirilir. Ancak Bulut tabanlı çözümler, artan iletişim maliyetleri ve gecikme süreleri nedeniyle her durumda ideal olmayabilir. Bu noktada, Sis bilişim ara katmanı, yerel ve zaman duyarlı karar alma işlemleri için uygun maliyetli ama yeterli performansa sahip çözümler gerektirir. Bu durum, görece daha hafif ve özelleştirilebilir işletim sistemlerine yönelimi artırmaktadır [13].

Linux'un Sis bilişim uygulamaları için tercih edilmemesinin temel nedenleri aşağıda özetlenmiştir:

- **Modülarite**: Linux'un monolitik çekirdek yapısı, Sis bilişim işlevlerinin doğrudan çekirdek alanına entegrasyonunu zorlaştırır. Mikro çekirdek mimarileri bu açıdan daha uygun bir yapı sunar [14]. Geniş API ve kütüphane desteğine rağmen, bu yapı karmaşıklığı artırmakta ve geliştirme süresini uzatmaktadır. Öte yandan, çekirdek düzeyinde çalışan yalın sistemler daha az bağımlılıkla geliştirilebilir.
- **Dayanıklılık**: Güvenlik açısından sistemin dayanıklı olması, açıkların hızla giderilebilmesi gerekir. Linux sağlam güvenlik mekanizmalarına sahip olsa da geniş kod

tabanı nedeniyle potansiyel açıklar barındırabilir ve bu açıkların yönetimi karmaşık olabilir [15]. Sistemden gereksiz bileşenlerin çıkarılması, saldırı yüzeyini azaltır ve güvenliği artırır.

- **Deterministik Davranış**: Tekil adres alanına sahip yalın işletim sistemleri, Linux'a kıyasla daha öngörülebilir ve belirlenebilir davranış sergiler. Gerçek zamanlı veya hassas zamanlamaya sahip uygulamalar için bu durum önemli bir avantaj sağlar.
- **Ölçeklenebilirlik**: Linux sistemlerinin çekirdek düzeyinde çalıştırılması mümkün olmadığından, kullanıcı alanı kütüphanelerine ihtiyaç duyulur. Bu da gömülü sistemlerde sistem bakımını ve ölçeklenebilirliği zorlaştırır. Buna karşılık, hiper yöneticiler aracılığıyla entegre edilen daha yalın çözümler, kaynakların yönetimini kolaylaştırır.
- **Verimlilik**: Linux, varsayılan olarak belirli bir kaynak tüketimi ile gelir ve her yeni işlem bu tüketimi artırır. Kaynak kısıtlı ortamlarda bu durum önemli bir dezavantaj oluşturur.

Yüksek performans ve düşük sistem ayak izine sahip çözümler için, kullanıcı alanında çalışan Linux uygulamaları yetersiz kalabilmektedir. Bu nedenle, bu çalışma OpenFog Konsorsiyumu'nun tanımladığı Sis bilişim hizmetlerinin doğrudan çekirdek alanına entegre edilmesi fikrini temel almaktadır. Geliştirilen çözümde, yalnızca temel çekirdek modülleri korunmuş, gereksiz kullanıcı alanı bileşenleri sistemden çıkarılmıştır. Böylece sistem, tekil adres alanı içinde sadeleştirilmiş ve hafifletilmiş bir yapıda çalışmaktadır.

Öte yandan, Sis bilişim sistemleri genellikle ARM mimarisi gibi gömülü sistemlerde yaygın olarak kullanılan mimariler üzerinde çalışmaktadır. Ancak ARM mimarisi altında farklı donanım yapılarına sahip çok sayıda özel konfigürasyon bulunmaktadır. Bu durum, mimari düzeyde üreticiler arasında ortak bir yapı geliştirilmesini zorlaştırmakta, donanım bağımlılığını artırmaktadır.

2.2 Sis Bilişim için RISC-V Mimarisi

Sis bilişim uygulamaları, gecikmeye duyarlı görevlerin ağın uç noktalarında gerçekleştirilmesini gerektirmektedir. Bu gereksinim, donanımsal düzeyde özelleştirilebilir ve verimli işlem yeteneklerine sahip mimarileri gerekli kılmaktadır. RISC-V, açık kaynaklı ve modüler yapısıyla bu gereksinimlere doğrudan yanıt veren potansiyel bir mimari olarak öne çıkmaktadır.

RISC-V'in açık ve esnek yapısı, geliştiricilere donanım üzerinde tam kontrol sağlarken, özelleştirilebilir komut kümeleri sayesinde farklı uygulama senaryolarına uyarlanabilir çözümler üretmeyi mümkün kılar [16]. Bu şeffaflık, Sis bilişim ve gömülü sistemler alanında daha küçük aktörlerin büyük üreticilere bağlı kalmaksızın özel sistemler geliştirmesine olanak tanır. Ayrıca mimari düzeyde yapılan değişikliklerin açık olarak izlenebilir olması, güvenlik ve doğrulama açısından önemli bir avantaj sunmaktadır.

RISC-V mimarisi, hem doğrudan donanım üzerinde çalışan sistemler (bare-metal) hem de sanallaştırma destekli çözümler için makine ve süpervizör kipleri aracılığıyla gerekli esnekliği sunmaktadır [17]. Bu nedenle, bu çalışmada geliştirilen PusOS işletim sistemi, açık kaynaklı RISC-V komut seti üzerine inşa edilmiştir. Yaygın şekilde benimsenen bu mimari ile edinilen deneyimin, gelecekteki donanım-yazılım bütünleşik çözümler için önemli katkılar sunacağı değerlendirilmektedir.

3. İlgili Çalışmalar

Giriş bölümünde de vurgulandığı üzere, mevcut akademik literatürde Sis bilişim genellikle Linux tabanlı platformlara özgü olarak yapılandırılmış çerçeve mimarilerle ele alınmaktadır. Bu bölümde, araştırma odağımızla doğrudan ilişkili işletim sistemi düzeyindeki çalışmalar ayrıntılı biçimde incelenmektedir. Özellikle BasatNet, FogOS, OpenStack ve FORA çözümleri, bu çalışma bağlamında değerlendirilmiştir.

Yazarların bir önceki çalışmasında, heterojen mimariye sahip cihazlara ölçeklenebilen, dağıtık bir bilişim ortamı olan **BasatNet** önerilmiştir [18]. Bu sistem sayesinde, kaynakları sınırlı düğümlerin kümelenmesiyle ortak bir çalışma ortamı sağlanmıştır. Bu yaklaşım, Sis bilişimi; yüksek başarımlı hesaplama (HPC) ve büyük veri sistemleri arasında konumlandıran minimalist bir hibrit model olarak değerlendirilebilir. Linux konteynerleri kullanılarak kaynak tüketiminin azaltılması hedeflenmiştir. Ancak, dağıtım ve yönetim zorlukları nedeniyle Sis bilişim katmanında Linux yerine özelleştirilmiş bir nano çekirdek kullanımına yönelinmiştir.

FogOS, Bulut, Sis ve Uç ortamlar arasında kaynak yönetimine odaklanan yeni bir işletim sistemi tasarımı olarak sunulmuştur. Sistem, kaynak ve uygulama yönetiminin yanı sıra uç cihazlar için kayıt, kimlik/adresleme ve kontrol arabirimlerini kapsayan bir referans mimari önermektedir. Ancak temel bileşenler ve çekirdek düzeyinde ayrıntılar sunulmamıştır. Bu da önerilen yapının klasik anlamda bir işletim sistemi değil, daha çok dağıtık bir uygulama çerçevesi olduğunu göstermektedir. Donanım ve temel işletim sistemi bileşenleriyle etkileşim konusundaki bu eksiklik, FogOS'u kavramsal düzeyde sınırlamakta ve "işletim sistemi" terimini mecazi kılmaktadır [19].

OpenStack, Stack4Things Nesnelerin İnterneti çerçevesi ve konteyner yönetimi altyapısı aracılığıyla bir Sis bilişim ortamı oluşturmayı hedeflemektedir. Yerel Sis ağı kullanılarak, Bulut ile olan veri iletimi hacminde azalma sağlandığı gösterilmiştir. Ancak sistem, Docker konteynerleri ve OpenStack altyapısına sıkı sıkıya bağlıdır. OpenStack'in sunduğu depolama ve hesaplama hizmetleriyle entegre yapısı, çözümü daha çok bir hizmet entegrasyon modülü konumuna getirmektedir. Dağıtım süreçlerinin bağımlılığı ise esneklik açısından sınırlayıcıdır [20].

FORA, Sis bilişimi Endüstri 4.0 uygulamaları bağlamında değerlendiren başka bir projedir. Geliştirilen Fog Computing Platform (FCP), görev açısından kritik kontrol uygulamalarını sanal ortamlarda izole biçimde çalıştırmaktadır. Her uygulama, kendi bölgesinde gerçek zamanlı bir işletim sistemine sahiptir. Bu platformda PikeOS gibi mikro çekirdek mimarileri tercih edilmiştir. Bu durum, bu çalışmada önerilen yapı ile örtüşmektedir; zira önerilen işletim sisteminde de

mikro çekirdek yaklaşımı benimsenmiş, ancak uygulamalar doğrudan çekirdeğe gömülerek hizmetlere doğrudan erişim sağlanmıştır [21].

Özetle, **FogOS** kavramsal bir dağıtık uygulama çerçevesi sunarken, **OpenStack** Linux temelli bir Bulut platformudur; **FORA** ise kontrol mühendisliğine odaklanmış endüstriyel bir Sis bilişim mimarisi önermektedir. Bu alandaki mevcut çalışmalarda, Sis bilişime özel bir işletim sisteminin eksikliği dikkat çekmektedir. Bu boşluğu doldurmak amacıyla, Sis işlevlerinin doğrudan nanokernel düzeyinde gerçekleştirildiği **PusOS** işletim sistemi bu çalışmada önerilmektedir. Bir sonraki bölümde, PusOS'un tasarım ve mimarisi ayrıntılı biçimde ele alınacaktır.

4. Tasarım ve Mimari

Önerilen işletim sisteminin mimarisinde alınan temel tasarım kararı, sistemin yalnızca çekirdek alanında çalışacak şekilde yapılandırılmasıdır. Bu yaklaşım, Şekil-1'de şematik olarak gösterilmiştir. Geleneksel işletim sistemlerinde, kullanıcı uygulamaları donanıma doğrudan erişemez; bu erişim, sistem çağrıları aracılığıyla çekirdek üzerinden dolaylı olarak sağlanır. Bu model, kullanıcıyı donanımdan izole etme avantajı sağlasa da işlem yapısı ve bağlam değiştirme gibi mekanizmalar nedeniyle önemli ölçüde bellek ve işlemci yüküne neden olabilir.

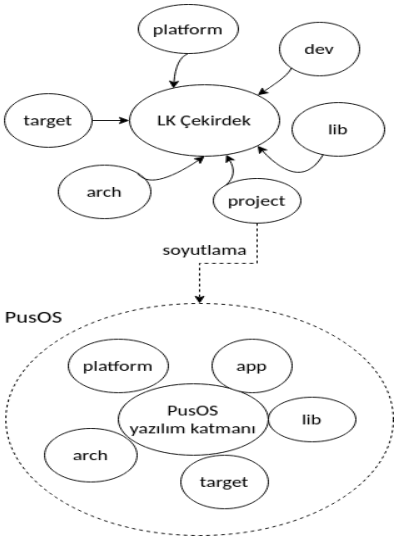
Sis bilişim ortamlarında ise, uygulamalar genellikle belirli ve sınırlı işlem kümeleri üzerine odaklanır. Bu özelleşmiş bağlamda, klasik kullanıcı alanı–çekirdek alanı ayrımı verimsizlik yaratmaktadır. Bu nedenle önerilen tasarımda, kullanıcı alanı tamamen ortadan kaldırılmıştır. İzolasyon ihtiyacı ise, RISC-V mimarisinin sunduğu ayrıcalık seviyeleri kullanılarak karşılanmaktadır.

Sistem, çalışma ortamına bağlı olarak çıplak donanım üzerinde ya da bir hipervizör altında, RISC-V mimarisinin sağladığı makine ya da süpervizör kiplerinde çalışmaktadır. Makine kipi, RISC-V mimarisinde en yüksek ayrıcalık seviyesine sahiptir ve bir RISC-V sisteminde zorunlu olan tek kiptir. Bu kipte çalışan kod, donanım bileşenlerine doğrudan erişebilir ve bu nedenle güvenilir kabul edilir. Önerilen işletim sistemi, tüm uygulamaları bu kipte tek bir adres alanı içerisinde çalıştırmaktadır. Bu yapı, işletim sisteminin hem çekirdek hem de uygulama görevlerini tek yetkili çalışma birimi olarak üstlenmesine olanak tanımaktadır.

Bu yaklaşım aşağıdaki avantajları beraberinde getirir:

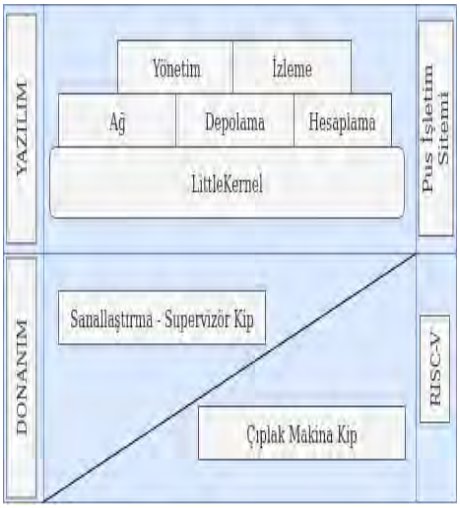
- Bağlam değiştirme (context switch) ihtiyacının ortadan kalkması
- Süreç yapısının kullanılmaması
- Kullanıcı alanı kütüphanelerinin gereksiz hale gelmesi
- Daha küçük kod tabanı ve düşük bellek kullanımı
- Uygulamaların doğrudan ve hızlı başlatılması
- Paylaşımlı adres alanı üzerinden verimli veri erişimi

Bu mimari, sis bilişimin gerektirdiği düşük gecikme ve yüksek verimlilik gibi özellikleri doğrudan donanım üzerinde sağlamayı hedeflemektedir. Bir sonraki bölümde, sistem bileşenlerinin işleyişi ve uygulama entegrasyon süreci detaylı olarak açıklanacaktır.



Şekil-1: PusOS'un LittleKernel üzerinde soyutlanan yapısı

Giriş bölümünde de belirtildiği üzere, Linux çekirdeğinde tüm işlevlerin tek bir çekirdek alanında gerçekleştirilmesi monolitik yapısından dolayı yüksek hata maliyetine sahiptir; çekirdek modüllerindeki bir hata tüm çekirdeğin çökmesine yol açabilir. Bu nedenle, Sis bilişim gereksinimlerini karşılamak amacıyla nanokernel tabanlı bir yaklaşım benimsenmiştir. PusOS'un temel çekirdeği olarak, ağ iç yapısı ve seri arayüz gibi temel işlevleri sağlayan Littlekernel projesi kullanılmaktadır [22]. Hesaplama, depolama, kontrol ve ağ işlevlerinden oluşan Sis bilişim bileşenleri ise bu çekirdek yapısı üzerine inşa edilmiştir. Mimari yapı Şekil-1'de gösterilmiştir.



Şekil-2: PusOS donanım üzerinde çalışması ve yazılım bileşenlerinin blok diyagramı

Şekil-2, PusOS'un bileşenlerini genel hatlarıyla sunmaktadır. PusOS, RISC-V donanımında iki farklı ayrıcalık kipinde çalışmaktadır. Her sistem gereksinimi, çekirdek modüllerinden biri olarak karşılanmaktadır. Geliştirme aşamasında bu modüller, soyut yazılım bileşenleri olarak ele alınmakta ve çekirdek derlenirken tüm modüller tek bir çekirdek dosyasında birleştirilmektedir. Çalışma zamanında ise bu modüller, çekirdek alanında ayrı adres alanları olarak

yer almakta ve paylaşılan bellek üzerinden birbirleriyle veri ve işlev paylaşımı gerçekleştirebilmektedirler.

Aşağıdaki bölümlerde, bu modüllerin Sis bilişim gereksinimlerine uygun olarak mimaride nasıl tasarlandığı detaylı şekilde açıklanacaktır.

4.1 Ağ

Sis bilişim, verilerin Bulut ile bağlantılı Nesnelerin İnterneti cihazları arasında ağ yığını üzerinden akmasıyla yoğun ağ etkileşimine dayanır. Katmana bağlı olarak farklı iletişim protokolleri kullanılsa da temel veri aktarımı genellikle TCP ve UDP taşıma protokolleri üzerine kuruludur. PusOS'ta ağ yığını, özellikle TCP ve UDP bağlantılarını destekleyecek şekilde tasarlanmış olup, CoAP ve MQTT gibi hafif protokoller aracılığıyla sensör verilerinin toplanmasını sağlar. İşlenen veriler, HTTP gibi protokoller vasıtasıyla Bulut'a güvenli bir şekilde iletilerek, IoT cihazları ile Bulut arasında kesintisiz ve güvenilir bir iletişim köprüsü oluşturulur.

Ayrıca, makine tarafından kolayca işlenebilen bir veri formatı olan Protocol Buffers, verimli veri kodlaması ve güçlü Enum türleri sayesinde ağ bant genişliğinin etkin kullanımı için öne çıkan bir standart olarak benimsenmiştir [23]. Sistem içinde, canlılık kontrol mekanizmaları ve uzak yordam çağrıları (RPC) gibi kritik iletişim işlevleri, mesaj serileştirme amacıyla Protocol Buffers kullanılarak gerçekleştirilmiş ve bu protokoller özellikle çekirdek alanında çalışacak şekilde optimize edilip doğrudan çekirdeğe entegre edilmiştir.

4.2 Depolama

Sis düğümü, sensör katmanından veri alır, işler ve gerektiğinde sonuçları Bulut'a iletir. Ayrıca, uygulama verilerinin yerel olarak Sis düğümünde saklanması gerekebilir. Veri saklama için en uygun ortam, dosya sistemi yapısıdır. Farklı dosya sistemleri mevcut olmakla birlikte, Sis bilişim uygulamaları için hata toleranslı ve güvenilir bir dosya sistemi kritik önem taşımaktadır. Ayrıca, PusOS yalnızca çekirdek alanında çalıştığından, gömülebilir ve çekirdek ile uyumlu bir dosya sistemi gereklidir.

Günlük bilgi işlem sistemlerinde Ext4, FAT, NTFS gibi yaygın dosya sistemleri kullanılırken, gömülü sistemler için optimize edilmiş çeşitli dosya sistemleri de mevcuttur. Sistemin tasarım hedefine uygun olarak, sınırlı kaynaklara sahip cihazlarda düşük kaynak tüketimi, yüksek performans, kritik veri güvenilirliği ve uygulamaya özgü esneklik gibi temel gereksinimleri karşılayan LittleFS dosya sistemi tercih edilmiştir [24]. Bu özellikleri sayesinde, Sis bilişim ortamında LittleFS dosya sistemi kullanılarak nano-çekirdeğe entegre bir yapı oluşturulmuştur. LittleFS'in davranışı standart dosya sistemlerine oldukça benzemekte ve gömülü sistemler için ideal bir çözüm sunmaktadır.

4.3 Hesaplama

Sis bilişim ortamında, veriler genellikle bağlı cihazlara yakın konumlandırılmış "Sis Düğümlerinde" işlenir; bu sayede verilerin sürekli olarak cihaz ile Bulut arasında iletilmesine gerek kalmaz. Bu yakınlık, düğümlerin gerekli verileri hızlıca sağlamasına ve temel görevleri etkin biçimde yerine getirmesine imkan tanır. İşlenen verilerden elde edilen sonuçlar daha sonra Bulut'a iletilir. Ancak, bu süreç dinamik değişimleri etkin şekilde yönetebilecek esnek bir hesaplama ortamı gerektirir. Kural motorları ve karmaşık olay işleme teknikleri belirli bir esneklik sağlamakla birlikte, artan karmaşıklık ek yazılım bileşenlerinin kullanımını zorunlu kılar. Bu nedenle, çekirdek alanı sınırları içinde kalarak daha basit ve düzenli bir çalışma ortamı sunan alternatif yaklaşımların incelenmesi önemlidir.

Bu yaklaşımı uygulayan en güncel çalışma olarak, Linux çekirdeğinde ağ, hata ayıklama, izleme ve güvenlik duvarı gibi işlevleri sunan eBPF projesi örnek gösterilebilir [25]. Yüklenip doğrulandıktan sonra eBPF programları, çekirdek içi anında derleme (just-in-time compiling) yöntemiyle yorumlanır veya derlenir ve böylece yerel yürütme performansı sağlar.

Bu çalışma için de benzer yaklaşım uygulanarak, çekirdek çalışırken program kodunun dinamik olarak oluşturulmasına ve yürütülmesine olanak veren özel bir betik dili (kaynak kod yolu: pus/exp/mu) geliştirilerek entegre edilmiştir. Bu dil kullanılarak dinamik çalışma zamanı ortamı, çekirdeğin hesaplama yeteneklerine yerleşik modüllerin yanında esnek bir güvenli betik yürütme mekanizma desteği sağlamaktadır. Betik dili ile ilgili temel özellikler ve teknik detaylara aşağıda yer verilmiştir:

- **Veri tipleri**: Betik dili, beş temel veri tipini destekler: Yok değer (nil), sayılar, metinler, tablolar ve fonksiyonlar. Bu sade yapı, düşük kaynaklı cihazlarda hızlı yürütmeyi ve bellek kullanımının optimize edilmesini sağlar.
- **Esnek tablolar**: Tablolar, anahtar/değer çiftleriyle esnek veri organizasyonu sunar ve herhangi bir veri tipi anahtar veya değer olarak kullanılabilir.
- **Fonksiyonellik**: Fonksiyonlar birinci sınıf nesne olarak işlev görür. İç kapsam ile kendi durumlarını saklayabilir ve başka fonksiyonlarla kolayca birleştirilebilir bu sayede modüler ve yeniden kullanılabilir betikler oluşturulabilir.
- **Global veri yönetimi**: Global yorumlayıcı kilidi mekanizması, kod içindeki global verilere senkronize erişimi garanti altına alır ve yarış durumlarını önler.
- **Veri işleme ve dağıtık hesaplama**: Betik dili, temel veri işleme işlevlerini (ör: map, filter, reduce) entegre ederek büyük veri kümelerinin toplu işlenmesine imkân tanır; bu özellik, gelen veri akışlarının daha verimli ve ölçeklenebilir şekilde işlenmesini sağlar.

Aşağıda Kod-1 kod bloğunda örnek bir Quicksort sıralama algoritmasının betik dilinde gerçekleme örneğine yer verilmiştir.

Kod-1: Quicksort Sıralama Algoritması Örnek Kodu

```
fn qsort(data)
  let [x, ..data] = tbl(data)
  if (len(data) == 0)
    return [x]

  let small = filter(fn(y) -> y <  x, data)
```

```
    let large = filter(fn(y) -> y >= x, data)
    return qsort(small) ++ [x] ++ qsort(large)
```

Sonuç olarak, geliştirilen betik dili entegrasyonu, sistemin düşük maliyetli cihazlarda güvenli ve esnek betik yürütmesini mümkün kılmaktadır. Bu yaklaşım, yazılım tabanlı güvenlik mekanizmalarının uygulanabilirliğini artırırken, çekirdeğin dinamik işleme kapasitesini de güçlendirmekte ve Sis düğümlerinde esnek, ölçeklenebilir bir çalışma ortamı sağlamaktadır.

4.4 Sanallaştırma

Sanallaştırma, kaynakların Bulut veri merkezlerinden bireysel Nesnelerin İnterneti cihazlarına kadar geniş bir yelpazede dağıtılmasını sağlayarak, Sis bilişim uygulamalarının ihtiyaçlarını karşılayacak şekilde özelleştirilmiş ve kesintisiz bir altyapı sunar. Soyutlama, orkestrasyon ve izolasyon mekanizmaları aracılığıyla, sanallaştırma; donanım, işletim sistemi, depolama aygıtları, ağ kaynakları ve olay işleme gibi sistem kaynaklarının sanal sürümlerinin oluşturulmasına imkân tanır [26]. Bu teknoloji aynı zamanda sistem güvenliğinin artırılmasında da kritik bir rol oynar.

Sanallaştırma teknolojileri arasında en yaygın olarak bilinenler hipervizör ve konteyner çözümleridir. Hipervizör, donanım soyutlama katmanında kaynakları sanallaştırarak, birden fazla sanal makinenin eş zamanlı olarak çalıştırılmasını mümkün kılar. Bu sanal makineler hipervizör aracılığıyla sanallaştırılmış donanım kaynaklarını paylaşarak kaynak kullanımının verimliliğini artırır [27]. Günümüzde Sis platformlarının uygulanacağı hemen tüm işlemci donanımları donanıma dayalı sanallaştırma mekanizmalarını desteklemektedir.

Girdi/Çıktı ve hesaplama kaynaklarının donanım tabanlı sanallaştırması, birden çok varlığın aynı fiziksel donanımı paylaşmasına olanak tanır. PusOS ise sanal makinelerden izole bir biçimde Sis düğümü işlevini yerine getirir. Böylece sanallaştırma, kaynak yönetimini kolaylaştırırken, sistemin belirli durumlarının yedeklenmesini ve gerektiğinde anlık görüntülerin geri yüklenmesini mümkün kılar. Ayrıca sanal makineler kaynak planlaması amacıyla başka ana bilgisayarlara taşınabilir.

Konteyner teknolojisi bazı avantajlar sağlamakla birlikte, göreceli yeniliği ve Linux'e özgü özelliklere olan bağımlılığı nedeniyle önerilen hafif hedef gereksinimlerle tam anlamıyla uyumlu olmayabilir. Kullanıcı alanı bileşenleri izolasyonu sunmasına rağmen, erişilmek istenen hafiflik ve yalın modelden sapmalar yaşanabilir.

4.5 Yönetim

Dağıtık sistemlerin yönetiminde esneklik, çağdaş bilişim altyapıları için vazgeçilmez bir paradigma haline gelmiştir. Özellikle Sis bilişim sistemleri bağlamında, donanım üzerinde doğrudan icra edilen veya sanallaştırılmış ortamlarda konuşlandırılan düğümlerin uzaktan yönetilebilirliği, operasyonel sürdürülebilirlik açısından kritik bir gereklilik arz etmektedir. Bu gereklilik, coğrafi olarak dağınık veya fiziksel erişimin kısıtlı olduğu düğümlerin etkin bir biçimde denetlenmesini ve işletilmesini mümkün kılmaktadır. Bir Pus düğümü, yüksek performanslı Uzaktan Yordam Çağrısı (RPC) mekanizmaları ve daha düşük seviyeli seri bağlantılar aracılığıyla yönetilebilir; RPC implementasyonlarında komutların hızlı ve tutarlı iletimi için sıklıkla Protocol Buffers kullanılmakta ve Pus düğümleri kendi aralarında da RPC üzerinden iş birliği yapabilmektedir. Sanallaştırılmış ortamlarda yönetim ise Linux tabanlı KVM platformları üzerinde, Libvirt yönetim kütüphanesi ve QEMU'nun Makine Protokolü (QMP) API'si ile desteklenmektedir; biz de PusOS deneyleri sırasında, ağ üzerinden iletişimin sağlanması ve çeşitli makine işlemlerinin yürütülmesi amacıyla sanal makineye ait olan QMP API'sini kullandık. QMP'nin, QEMU sanal makineleriyle düşük seviyeli etkileşim kurmak için güçlü bir mekanizma sağlayan nitelikleri uzaktan yönetim senaryolarında kritik bir rol oynamaktadır. Tüm bu kapsamlı uzaktan yönetim yetenekleri, Sis bilişim sistemlerinin operasyonel verimliliğini maksimize ederek, büyük ölçekli ve coğrafi olarak dağıtık Nesnelerin İnterneti ve uç bilişim uygulamalarının kesintisiz çalışmasını temin etmektedir.

4.6 İzleme

Dağıtık yapısı ve potansiyel kör noktaları nedeniyle, Sis bilişim sistemlerinin izlenmesi, geleneksel Bulut sistemlerine kıyasla daha fazla zorluk barındırmaktadır. Ancak etkili bir izleme mekanizması, sistemin doğru düzenlenmesi ve yönetimi için hayati önem taşır. Bu nedenle, Sis düğümlerinin ve iletişim bağlantılarının durumu ile çalışan görevlerine ilişkin güncel bilgilerin toplanması ve orkestrasyonu gereklidir. Servislerin daha güçlü düğümlere aktarılması ve yerleşim optimizasyonlarının geçmiş verilere dayandırılarak gerçekleştirilmesi, hizmet düzeyi anlaşmalarının sağlanmasına ve sistemin kesintisiz çalışmasına önemli katkılar sunar [30].

İzlenen düğümlerin heterojenliği yanında, izleme faaliyetleri sıklık, topoloji ve iletişim örüntülerine ilişkin durumları da kapsar. Sis düğümünde çalışan sistemin sağlık durumu ve görevleri yerine getirme performansı da izlenmek istenen diğer kritik parametrelerdendir. Karar alma süreçlerine veri sağlamak amacıyla, PusOS gerçek zamanlı sistem izleme yetenekleri sunmaktadır. TCP/IP protokollerini destekleyen düğümler, HTTP protokolü üzerinden canlı sistem durumu ve işlem gözlemi için erişim imkânı sağlar; ayrıca, ayrıntılı sistem bilgilerine çeşitli API'ler aracılığıyla ulaşılabilmektedir. Bu veri kanalları, kapsamlı kayıt tutma işlemlerini gerçekleştirmek üzere günlük sunucuları ile entegre edilebilmektedir.

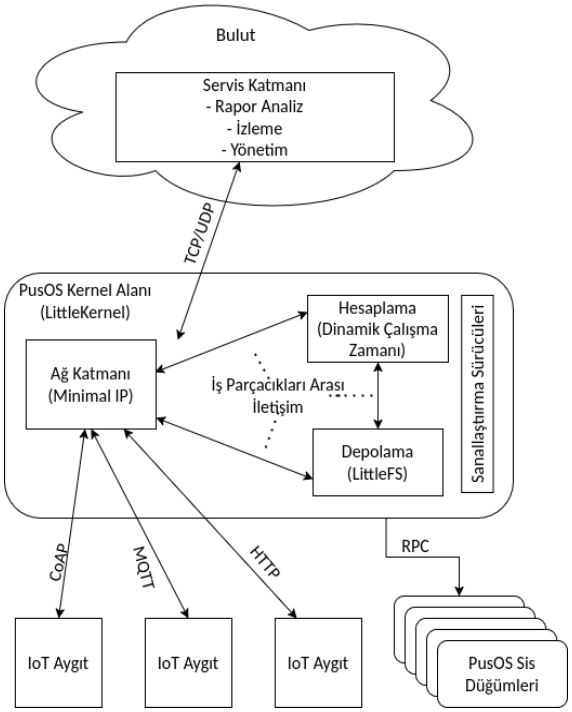
4.7 Dağıtım

Uygulama geliştirme süreci, hedef sistem üzerinde çalışacak tüm yazılım bileşenlerinin eksiksiz olarak dahil edilmesini gerektirir. En basit Python betikleri dahi çalıştırılabilmek için yorumlayıcı ve kütüphane desteğine ihtiyaç duyar. Linux işletim sistemlerinde uygulama karmaşıklığı arttıkça, yazılımlar genellikle paket yapıları altında sunulmaktadır. Daha özel ve kısıtlı kaynaklı projelerde ise, Docker benzeri konteyner teknolojileriyle yazılım dağıtımı mümkün olmakla birlikte, bu yaklaşımlar daha ağır sistem gereksinimlerine sahiptir.

Sis bilişim uygulamaları, veriyi işleyip Nesnelerin İnterneti cihazlarına yakın konumda hızlı kontrol kararları almak üzere

tasarlanmıştır; buna karşılık, geleneksel sunucu uygulamaları veriyi merkezi bir yerde işler ve karar verir. Çoğu Nesnelerin İnterneti uygulaması, sensör verisi toplama, temel kontrol kararları verme ve ilgili cihazları yönetme görevlerini üstlenir. Zaman zaman, özetlenmiş sensör verileri daha ileri analiz için Bulut'a iletilmekte, bu da kontrol mantığının hiper-parametrelerinin Bulut üzerinden ayarlanmasına imkân sağlamaktadır. Sis bilişim perspektifinden bakıldığında, Nesnelerin İnterneti düğümlerinde gerçekleştirilen işlemler minimal ve iyi tanımlanmıştır.

Bu bağlamda, uygulama mantığının tamamını tek bir çekirdek içinde bütünleştirerek dağıtım maliyetlerini azaltmayı hedeflemekteyiz. Böylece, doğrudan hedef donanım üzerinde çalıştırılabilen birleşik bir çekirdek imajı elde edilmektedir. Bu yaklaşım, Linux sistemlerinde karşılaşılan çoklu yazılım bileşenlerinin yönetilmesi ihtiyacını ortadan kaldırarak gereksinim planlamasını basitleştirir. Tüm kodu verimli biçimde, tek bir çekirdek çözümünde paketleyerek, alanlara özgü ihtiyaçlara özel optimize edilmiş bir dağıtım gerçekleştirilmektedir.



Şekil-3: Bulut-Nesnelerin İnterneti entegrasyonunda PusOS yerleşimi

Şekil-3'te gösterildiği üzere, PusOS, Bulut bilişim katmanından Nesnelerin İnterneti katmanı cihazlarına yönelik etkileşim sürecini koordine eden bir aracı olarak konumlanmaktadır. Bu mimaride, PusOS, ağ iletişimi, hesaplama kaynakları yönetimi ve veri depolama işlevlerini kapsayan temel çekirdek modülleri ile donatılmıştır. PusOS'un mimarisi, bulut katmanı ile IoT katmanı arasındaki kesintisiz iletişimi sağlayan miniIP ağ katmanı, çekirdek modülleri etrafında şekillenmiştir. Bu modüller, veri paketlerinin yönlendirilmesi ve protokol dönüşümlerinin gerçekleştirilmesi gibi ağ işlevlerini üstlenerek, farklı katmanlar arasında güvenilir bir bağlantı köprüsü kurar. Ağdan gelen verilerin işlenmesi ve depolanması süreçleri, işletim sisteminin çekirdek modülleri olarak gerçeklenmiş çekirdek modülleri tarafından yürütülmektedir. Bu modüller, performans optimizasyonu amacıyla paylaşılan bellek (shared memory) mekanizması üzerinden iş parçacıkları (threads) seviyesinde yüksek hızlı ve düşük gecikmeli bir iletişim kurabilmektedir. Bu iletişim mekanizması, veri akışının verimli bir şekilde yönetilmesine ve iş yüklerinin paralel olarak işlenmesine olanak tanır. Diğer Sis düğümleriyle arasındaki etkileşim, iki ana mekanizma aracılığıyla gerçekleştirilir: İlk olarak, Protocol Buffers kullanılarak geliştirilmiş özel bir uzak yordam mekanizması devreye girer. Bu mekanizma, düğümler arası iş birliği ve komut iletimi için standartlaştırılmış, yüksek verimli bir arayüz sağlar. İkinci olarak, sanallaştırma birimi ile doğrudan iletişim kuran özel bir servis bulunmaktadır. Bu servis, sanallaştırılmış ortamdaki kaynakların yönetimi ve kontrolü için hayati öneme sahiptir.

5. Deneysel Çalışma ve Değerlendirme

Bu bölümde, PusOS'un çekirdek alanı gömülü yapısı ve sanallaştırma yeteneklerini değerlendirmek amacıyla yapılan testler sunulmuştur. Bu işlevsellikleri değerlendirmek için, çeşitli mimarileri (RISC-V dahil) destekleyen yüksek performanslı bir sanal makine olan QEMU (sürüm 9.0.1) emülatörü kullanılmıştır [31]. Ana işletim sistemi olarak Milis Linux tercih edilmiştir [32]. Çekirdek, GCC (sürüm 13.2.0) C ve C++ derleyicisi kullanılarak derlenmiştir.

Karşılaştırma amacıyla, geliştirme araçlarıyla zenginleştirilmiş popüler bir dağıtım olan Fedora Linux (sürüm: 38-20230519-SiFive-QEMU) seçilmiştir. RISC-V mimarisi ve QEMU sanal makine desteği gereksinimleri nedeniyle, mevcut hazır Fedora Linux imajı kullanılmıştır. Testlerde yalnızca çekirdek ve ilgili kullanıcı alanı düzeyindeki performans ölçümlerine odaklanılmıştır ve yalnızca çekirdek sistem çağrıları üzerinden veri gönderme ve depolama işlemleri ölçülmüştür. Bu yaklaşımla, kullanıcı alanı servislerinden kaynaklanan ek yüklerin ölçüm sonuçlarını etkilemesini önlenmiştir.

Test uygulamasında kullanılan iletim kütüphanesi ve diğer standart işlevler, LibC paylaşımlı kütüphanesi kullandığı için ölçümlerde Linux tarafında zorunlu olarak kullanıcı alanına da ihtiyaç duyulmaktadır. Ayrıca, çalışmanın amacı kullanıcı alanının maliyetini de göstermek olduğundan, kullanıcı alanının dahil edilmesi kasıtlı bir tercih olmuştur. Bu yapılandırma, PusOS ile Linux çekirdeği arasındaki sistem çağrısı ve çekirdek işlevselliği performansını doğrudan ve tutarlı bir biçimde karşılaştırmayı mümkün kılmaktadır.

Tek bir ELF dosyası olan PusOS, QEMU sanal makinesine çekirdek parametresi olarak verilmiştir. Aşağıdaki Çizelge-1'de, testlerde kullanılan PusOS ve Linux sistemlerinin özelliklerini karşılaştırmalı olarak göstermektedir.

Çizelge-1: Test sistemlerinin Karşılaştırılması

Özellik	PusOS	Fedora Linux
Çekirdek	LittleKernel	Linux
Çekirdek Tipi	Nanokernel	Monolitik
Süreç Yapısı	Çekirdek İş Parçacığı	Kullanıcı Süreci

Asgari Bellek Kullanımı	40	150
Dağıtım Metotu	ELF	IMG

Aşağıdaki alt bölümler, sırasıyla veri toplama, depolama yönetimi, hata kurtarma ve uygulama göçü başlıklarıyla operasyonel ayrıntıları sunmaktadır.

5.1 Veri Toplama

Bu deneyde kullanılmak üzere 4 baytlık Float tipindeki sanal duyarga verisi üreten bir mekanizma tasarlandı. Test sistemlerine yerleştirilen istemci uygulaması, CoAP protokolü kullanarak duyarga verilerini çekmektedir. Algoritma-1'deki SAYAC parametresiyle sınırlandırılmış bir dizi test gerçekleştirilmiş ve sensör verisi çekme performansı PusOS ile Linux sistemleri arasında karşılaştırılmıştır.

Tipik bir Nesnelerin İnterneti ağ kurulumunu taklit etmek amacıyla tasarlanan Algoritma 1, sensör okuma işlemlerini iş parçacıkları kullanarak eşzamansız şekilde yürütmektedir. Hazırlanan simülasyonun temel amacı, çekirdek düzeyindeki sistem çağrılarının işletim sistemleri üzerindeki yükünü ölçmektir.

Algoritma Girdileri: İki temel parametre alır: GECIKME (her bir okuma-yazma işlemi arasındaki gecikme süresini milisaniye cinsinden belirler, örnek: 10, 50, 100 ms) ve SAYAC (her bir iş parçacığının kaç kez okuma/yazma işlemi yapacağını belirleyen işlem sayısı).

Algoritma Çıktıları: Algoritmanın çıktıları, deneysel ölçümlerin ana sonuçlarını oluşturur. Bunlar; tamamlanan işlem sayısı, uygulamanın bu işlemleri tamamlamak için harcadığı toplam süre ve Denklem (1)'de gösterildiği üzere işlem başına düşen ortalama mesaj işleme süresi (OMİS)'dir.

$$OMİS = S_{toplam} - İ_{sayı} \quad (1)$$

- S_{toplam} : Uygulamanın tüm işlemleri tamamlamak için harcadığı toplam süredir.
- $İ_{sayı}$: Uygulama tarafından başarıyla tamamlanan toplam işlem sayısıdır.

Algoritma 1'deki GOREV fonksiyonu, belirlenen GECIKME süresiyle simüle edilmiş bir sensör okuma ve yazdırma işlemini, SAYAC sayısı kadar tekrarlı bir döngüde gerçekleştirir. Ana iş parçacığı, alt iş parçacıklarının yaşam döngüsünü yönetmek için create, resume ve join gibi iş parçacığı ilkel fonksiyonlarını kullanır. Gerçek dünyadaki eşzamansız iletişim paradigmalarının aksine, bu kasıtlı olarak eşzamanlı iş parçacığı yönetimi, farklı işletim sistemlerinde çekirdek düzeyinde meydana gelen bağlam değiştirme (context switching) ve sistem çağrısı yüklerinin net bir şekilde karşılaştırılmasını sağlar. Tablo-2'de sunulan veriler, 10 kez tekrarlanan testlerin ortalama değerlerini içermektedir.

Algoritma-1: Veri Toplama ve Depolama

```
1:  Thread (SAYAC, GECIKME):
2:    for i = 0 to SAYAC
3:      τ = create_thread(&TASK)
4:      resume_thread(τ)
5:      join_thread(τ)
6:      delay (GECIKME)

7:  GOREV ():
8:      Val = readSensor()
9:      print (Val)
10:     IF DEPOLAMA_AKTIF:
11:       f = Open_text_file()
12:       Append_to_file(f, Val)
13:       Close_file(f)
```

Çizelge-2'de, PusOS üzerinde çalışan uygulamanın Linux muadilinden daha hızlı tamamlandığını göstermektedir. Bu performans avantajı, PusOS'un çekirdek alanında çalışmasından kaynaklanmakta ve böylece ek yük (overhead) en aza indirilmektedir. Alınan sensör değeri sayısı arttıkça, işlem süreleri arasındaki fark daha belirgin hale gelmektedir. PusOS, farklı gecikme sürelerinde (0,7-0,9 ms arasında) bir mesaj yakalayıp yazdırırken, Linux uygulamasında bu süre 3,4-4 ms olarak kaydedilmiştir. PusOS'ta sistem çağrısı olmadığı için, gecikme sürelerindeki değişiklik PusOS uygulamasını etkilememiştir.

Çizelge-2: Duyarga İletişim Hızları

		Pus OS		Linux	
Gecikme	**Mesaj Sayısı**	**Toplam (s)**	**OMİS (ms)**	**Toplam (s)**	**OMİS (ms)**
10	1000	10,80	0,80	13,52	3,52
10	2000	21,74	0,87	28,71	3,69
10	3000	32,60	0,86	41,05	3,68
50	1000	50,70	0,70	53,40	3,40
50	2000	101,37	0,69	106,82	3,41
50	3000	152,06	0,68	160,26	3,42
100	1000	100,70	0,70	103,95	3,95
100	2000	201,44	0,72	207,76	3,88
100	3000	302,20	0,73	311,70	3,90

Ancak Linux'ta gecikme azaldıkça, mesaj işleme süresi de azalmaktadır. Bunun sebebi, Linux çekirdeğinin yakın zamanda erişilen sayfaları takip edip aktif listede tutması ve son erişilen sayfaları listenin en üstüne yerleştirmesidir. Sonuç olarak, farklı sonraki mesaj bekleme gecikme süreleri (GECIKME) için PusOS ve Linux arasındaki ortalama mesaj işleme süresi (OMİS) karşılaştırıldığında, PusOS üstün performans göstermekte ve Linux'tan 4,86 kata kadar daha hızlı çalışmaktadır.

5.2 Depolama

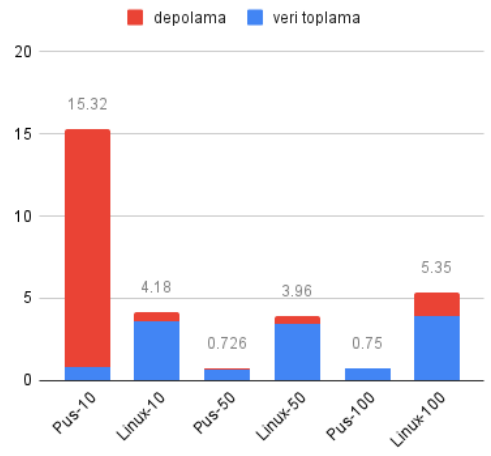
Bu bölüm, dosya sistemi ile ilgili G/Ç iş yükünün değerlendirilmesini kapsamaktadır. Sensör okumalarından elde edilen değerlerin işlenmesi ve seri iletişim yoluyla aktarılmasının yanı sıra, sensör verilerinin dosyaya kaydedilmesi ek bir görev olarak dahil edilmiştir. PusOS,

sensör verilerinin depolanmasını LittleFS modülü aracılığıyla yönetirken, Linux karşılaştırma amacıyla Ext4 dosya sisteminde bir dosyaya kaydetmek için bir istemci uygulaması kullanmaktadır. Her iki sistem de Qcow2 formatında QEMU blok disk kullanmaktadır. Her sensör okumasında, her kayıt için dosya açma/kapatma işleminin yürütülmesi test edilmiştir. Her okuma işlemi sırasında dosya açılır, değer yazılır ve son olarak dosya kapatılır. Detaylar Algoritma-1'de verilmiştir. Bu deneyde, bu işlemler sırasıyla 10, 50 ve 100 ms aralıklarla gerçekleştirilmiştir. Amaç, PusOS'un G/Ç işlem hızını Linux ana sistem ile karşılaştırmaktır. Karşılaştırmalı değerler Çizelge-3'te sunulmuştur.

Çizelge-3'te, LittleFS ile 50 ms ve 100 ms aralıklarında olağanüstü hızlı işlem süreleri elde edilmiştir. 10 ms aralığı hariç, PusOS Test-2 görevini, Test-1'e kıyasla mesaj başına yalnızca 0,1–0,5 ms daha yavaş tamamlamaktadır. Bu durum, dosya açma, yazma ve kapama işlemleri sırasında ortaya çıkan gecikmenin neredeyse ihmal edilebilir olduğunu göstermektedir.

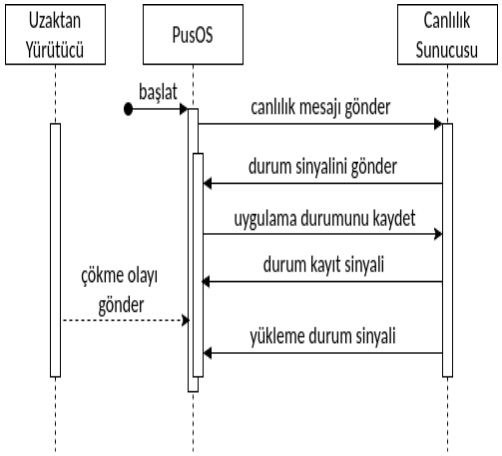
Çizelge-3: G/Ç İşlem Hızları

		Pus OS		Linux	
Gecikme	**Mesaj Sayısı**	**Toplam (s)**	**OMİS (ms)**	**Toplam (s)**	**OMİS (ms)**
10	1000	25,38	15,38	14,05	4,05
10	2000	48,03	14,01	28,24	4,12
10	3000	75,82	15,27	42,54	04,18
50	1000	50,74	0,74	54,07	4,07
50	2000	101,42	0,71	107,84	3,92
50	3000	152,19	0,73	161,97	3,99
100	1000	100,73	0,73	105,20	5,20
100	2000	201,47	0,74	210,70	5,35
100	3000	302,32	0,77	315,24	5,08



Şekil-4: Veri toplama ve depolama iş yükü karşılaştırması

10 ms için ise Linux, PusOS'tan 3,6 kata kadar daha hızlı dosya G/Ç işlemi gerçekleştirmektedir. Bunun sebebi, LittleFS için 10 ms altındaki gecikme değerlerinin hata içerdiğinin raporlanmasıdır [33]. Sonuç olarak, 10 ms üzerindeki gecikmelerde PusOS, ortalama mesaj işleme süresinde (OMİS) Linux'u geçmektedir. Çizelge-2 ve Çizelge-3'te yer alan değerler üzerinden veri toplama ve depolama arasındaki iş yükünün değerlendirildiği Şekil-4'te karşılaştırmalı olarak gösterildiği üzere, 50 ve 100 ms gecikme süreleri arasındaki ortalama değer açısından PusOS, Linux rakibinden 5,5 kata kadar daha hızlı çalışmaktadır.



Şekil-5: Hata kurtarma işlem akışı

5.3 Hata Kurtarma

Bu bölümde, PusOS'un bir çökme sonrasında kendini toparlayıp normal çalışmaya devam etme yeteneği değerlendirilmektedir. Deneyin akışı, süreçlerin daha iyi anlaması için aşağıda maddeler halinde özetlenmiştir:

1. Uygulama başlatma: Sistem açılışının ardından, seri porta Fibonacci serisini yazdıran basit bir uygulama başlatılır.
2. Anlık görüntü kaydı: Uygulama çalışırken, her bir Fibonacci sayısı yazıldıktan sonra 1 saniyelik gecikmeyle hesaplamanın anlık görüntüsü alınır.
3. Kasıtlı çökertme: Uygulama çalışmaya devam ederken, sistem uzaktan yönetim API'si kullanılarak kasıtlı olarak çökertilir.
4. Kurtarma işlemi: Çökme sonrasında, PusOS en son başarılı anlık görüntüden bağımsız olarak kendini otomatik olarak geri yükler. Bu işlem, QEMU'nun QMP API'si aracılığıyla gerçekleştirilir.
5. Göreve devam: Sistem, uygulamayı kaldığı noktadan kesintisiz bir şekilde devam ettirir ve seri porta Fibonacci sayılarını yazdırmaya devam eder.

Deney, içinde Fibonacci serisi uygulaması bulunan ve her biri ayrı bir sis düğümü olarak çalışan 10 sanal makine (SM) üzerinde PusOS ile gerçekleştirilmiştir. Dört başarısızlık test durumu vardır:

1. Rastgele 1 SM'nin çökertilmesi
2. Rastgele 3 SM'nin çökertilmesi
3. Rastgele 5 SM'nin çökertilmesi
4. Tüm 10 SM'nin (hepsi) çökertilmesi

Her test durumunun tutarlı ortalama zaman değerleri sağlaması için 10 kez tekrarlanmıştır. Sistem, her 1 saniyede bir bir kalp atışı mesajı göndererek arızayı tespit edecek şekilde yapılandırılmıştır. Ayrıca, her Fibonacci sayısı yazıldıktan sonra 1 saniyelik gecikmeyle hesaplamanın anlık görüntüsü alınmaktadır. Sistem üzerinde çalışan blok disk

kayıt altına alınmıştır. Kurtarma işlemi bu anlık görüntü kullanılarak gerçekleştirilir. Mevcut sistem durumunun kaydedilmesi ve geri yüklenmesi QEMU'nun QMP API'si aracılığıyla sağlanmaktadır. Hata tolerans mekanizmasının işleyişi Şekil-5'te detaylı olarak açıklanmıştır.

Çizelge-4: Hata kurtarma testi

Başarısız olma durumu	Ortalama kurtarma süresi (s)	Kurtarma gecikme süresi (s)
1 SM	43	1,91
3 SM	37,2	1,82
5 SM	33,6	1,83
10 SM (Hepsi)	26,5	1,65

Deneylerimizde kullanılan temel hata kurtarma metrikleri olan Ortalama Kurtarma Süresi ve Kurtarma Gecikme Süresi, sistemin bir çökme sonrası ne kadar hızlı bir şekilde eski kararlı durumuna dönebildiğini nicel olarak belirlemektedir. Bu metrikler, sistemin güvenilirliğini ve hata tolerans kapasitesini değerlendirmek için kritik öneme sahiptir ve aşağıdaki ayrıntılı şekilde açıklanmaktadır;

Ortalama Kurtarma Süresi (OKS), Denklem (2)'de gösterildiği üzere sistemin çökme anından itibaren tamamen işlevsel hale gelip, ilk başarılı kalp atışı mesajını gönderdiği ana kadar geçen toplam süredir. Bu süre, hem sanal makineyi son başarılı durumdan geri yükleme işlemini hem de uygulamanın tekrar çalışmaya başlaması için geçen süreyi kapsar.

$$OKS = T_{kalp\ atışı} - T_{çökme} \quad (2)$$

- $T_{kalp\ atışı}$: Sistem çöktükten sonraki ilk başarılı kalp atışı mesajının gönderildiği andır.
- $T_{çökme}$: Sistemin kasıtlı olarak çökertildiği andır.

Kurtarma Gecikme Süresi (KGS), Denklem (3)'de gösterildiği üzere sanal makinenin son başarılı anlık görüntüden yüklendiği andan, sistemin normal çalışmaya dönerek ilk kalp atışı mesajını gönderdiği ana kadar geçen süredir. Bu metrik, sistemin kurtarma sürecinin ne kadar sürede tamamlandığına odaklanır. Bu süre, 1 saniyelik Fibonacci serisi yazdırma gecikmesini de içerdiği için yükleme süresi ile ilk kalp atışı mesajı arasındaki farkı temsil eder.

$$KGS = T_{kalp\ atışı} - T_{yükleme} \quad (3)$$

- $T_{kalp\ atışı}$: Sistemin yeniden yüklendikten sonra ilk kalp atışı mesajını gönderdiği andır.
- $T_{yükleme}$: Çökmüş sanal makineye yükleme sinyali gönderildiği ve anlık görüntünün geri yüklenmeye başladığı andır.

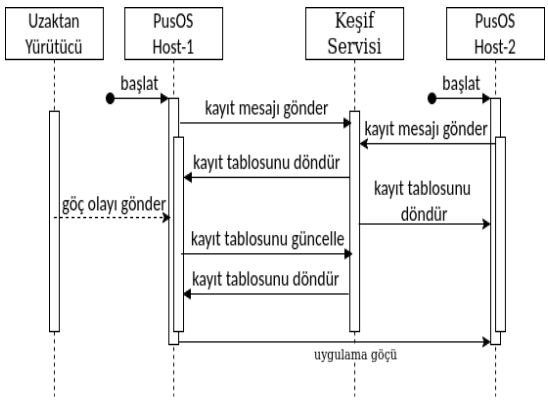
Çizelge-4'te belirtilen bulgulara göre, sistem uygulamayı son çalıştırıldığı noktadan başarıyla devam ettirmiş ve yükleme gecikmesi 2 saniyenin altında değerlerde gerçekleşmiştir. Ayrıca, kurtarma süresine sayı yazdırılırken oluşan 1 saniyelik gecikme dahildir. Bu süre, kurtarma gecikme süresi olarak değerlendirilir; çünkü çökmüş bir PusOS'un ana makinesine yükleme sinyali gönderildiğinde, sanal makine son başarılı anlık görüntüyü yükler. Ardından sistem çalışmaya devam eder ve kontrolöre kalp atışı mesajı gönderir. Bu nedenle, kurtarma gecikme süresi, yükleme süresi ile yüklemeden sonraki ilk kalp atışı mesajı arasındaki farktır.

Burada kullanılan hata kurtarma mekanizmasının temelinde QEMU/KVM hipervizör yetenekleri bulunmaktadır. Ancak asıl katkı, PusOS'un bu hipervizör tabanlı süreçlerle sorunsuz bütünleşerek kendi hata tolerans mekanizmalarını işlevsel biçimde destekleyebildiğini göstermektir. Böylece PusOS'un, mevcut sanallaştırma altyapılarının sunduğu kurtarma senaryolarına uyum sağlayabildiği doğrulanmıştır.

5.4 Göç

Bu deneyde, Sis bilişim sistemlerinde yük dengeleme amacıyla da kullanılan uygulama göçü özelliği sergilenmektedir. Daha önceki çökme kurtarma deneyine benzer şekilde basit bir Fibonacci serisi yazdırma uygulaması kullanılmıştır. Kurulum, QEMU kullanarak ayrı ana sistemlerde çalışan birden fazla Pus düğümünü içermektedir. Düğümlerin birbirlerinin durumunu kayıt tablosu aracılığıyla öğrenebilmesini sağlayan bir keşif servisi geliştirilmiştir. Deneyin akışını netleştirmek için, süreç aşağıdaki adımlarla özetlenmiştir:

1. Keşif servisi kaydı: Başlangıçta her bir Pus düğümü, birbirlerinin durumunu öğrenebilmek için Keşif Servisi'ne kaydolur ve en güncel kayıt tablosunu alır.
2. Göçün başlatılması: Düğümler, bu kayıt tablosuna dayanarak uygulama göçü olaylarını kendi kendine planlar. Uzaktan erişim ile bir uzaktan yürütücü, kasıtlı olarak bir göç olayı gönderir.
3. Hedefe göç: Şekil-6'da gösterildiği gibi, Düğüm-1 Keşif Servisi'nden kayıt tablosunu sorgular ve aldığı yanıta göre uygulamayı doğrudan Düğüm-2'nin çalıştığı hedef ana sisteme QEMU API aracılığıyla göç ettirir.
4. Uygulamanın devamı: Göç edilen yeni ana makine, uygulamanın kaldığı yerden devam etmesini sağlar. Bu deneyde kullanılan basit Fibonacci serisi uygulaması, seri çıkışa sayıları 1 saniyelik aralıklarla yazdırmaya devam eder.



Şekil-6: Uygulama göçü işlem akışı

PusOS'un göç sırasındaki performansını değerlendirmek için, göçün başlamasından normal işlemlerin tekrar başlamasına kadar geçen süre ölçüldü. Deneylerin 10 kez tekrarlanarak elde edilen ortalama değerlerin yer aldığı Çizelge-5.1'te gösterildiği gibi, PusOS sistem performansı üzerinde sürekli olarak minimum etki gösterdi. Göç nedeniyle beklenen yürütme süresinden sapmayı temsil eden ortalama gecikme

süresi sadece 0,25 saniye olarak kaydedildi. Bu, PusOS'un başarılı bir şekilde göçü gerçekleştirip operasyonları minimum kesinti ile kaldığı yerden devam ettirebildiğini göstermektedir. Çizelge-5.2'deki bulgulara göre ise, sistem 10 adet Fibonacci sayısını başarıyla göç ettirmiş, operasyonları 2 saniyenin altında bir değerle yeniden başlatabilmiştir. Bu durum, göçün toplam yürütme süresi üzerindeki etkisinin ihmal edilebilir düzeyde olduğunu göstermektedir.

Çizelge-5.1: Uygulama göç süresi performansları (saniye cinsinden)

Gecikme (s)	Seri Yazma (s)	Seri Yazma + Göç (s)	Saf Göç (s)
4	4,006	4,139	0,133
5	5,006	5,125	0,119
6	6,009	6,293	0,284
7	7,008	7,148	0,140

Çizelge-5.2: Fibonacci deneyi göç süresi değerlendirme (saniye cinsinden)

Durum	Seri Yazma (s)	Seri Yazma + Göç (s)	Saf Göç (s)
Tüm sayılar	9,008	10,961	1,953

Sonuçlar, PusOS'un canlı göç sürecinde yüksek verimlilik ve güvenilirlik sağlayabildiğini göstermektedir. Göç sırasında durma süresinin minimize edilmesi ve sistem performansının korunması, PusOS'u minimum kesinti gerektiren sis uygulamaları için uygun bir gömülü işletim sistemi haline getirmektedir. Her ne kadar göç işlemi QEMU/KVM hipervizör yetenekleri ile mümkün kılınsa da bu deneydeki temel katkı, PusOS'un söz konusu hipervizör mekanizmalarını herhangi bir uyumsuzluk veya ek yük olmaksızın desteklediğini ortaya koymaktır. Böylece PusOS'un sanallaştırma ortamlarında güvenilir çalıştığı ve mevcut altyapılarla bütünleşmeye elverişli olduğu doğrulanmıştır.

6. Tartışma

Bu çalışmada önerilen mimari, uygulamaların doğrudan çekirdek alanında çalıştırılmasına dayanmaktadır. Bu yaklaşım, bağlam değiştirme ve sistem çağrısı yüklerini ortadan kaldırarak performans ve verimlilik açısından önemli kazanımlar sağlamaktadır. Bununla birlikte, kullanıcı ve çekirdek alanı ayrımının ortadan kalkması güvenlik ve kararlılık bakımından çeşitli riskler barındırmaktadır. Geleneksel işletim sistemlerinde bu ayrımın temel işlevi, hatalı veya kötü niyetli bir uygulamanın çekirdeğe zarar vermesini ve dolayısıyla tüm sistemin çökmesini engellemektir. Önerilen mimaride ise bu izolasyon katmanının bulunmaması, tekil bir uygulama hatasının tüm sistemi tehlikeye atma olasılığını artırmaktadır.

Bununla birlikte, bu tasarım seçimi gömülü sistem ekosisteminde yaygın olarak benimsenen bir yaklaşıma karşılık gelmektedir. Örneğin, Arduino benzeri mikrodenetleyici tabanlı cihazlarda uygulamalar doğrudan tekil bir çekirdek üzerinde çalıştırılmakta ve kullanıcı–çekirdek ayrımı bulunmamaktadır. PusOS'un tekil çekirdek yapısı da benzer bir mantıkla çalışarak kaynak kısıtlı cihazlarda verimliliği artırmayı hedefleyen yalın bir model sunmaktadır. Dolayısıyla önerilen mimari yalnızca deneysel bir paradigma değil, aynı zamanda endüstride uzun süredir uygulanan gömülü sistem tasarımlarının devamı niteliğindedir.

Bu bağlamda, mimarinin potansiyel güvenlik zafiyetlerini azaltmaya yönelik aşağıdaki önlemler ve değerlendirmeler öne çıkmaktadır:

- **Nanokernel Yaklaşımı ve Minimal Kod Tabanı:** PusOS, Linux'un monolitik yapısından farklı olarak küçük bir kod tabanına sahip LittleKernel projesi üzerine inşa edilmiştir. Minimalist yapı, gereksiz bileşenlerin ortadan kaldırılmasıyla saldırı yüzeyini küçültmekte ve potansiyel hata veya güvenlik açıklarının sayısını azaltmaktadır.
- **RISC-V Ayrıcalık Seviyeleri:** Uygulamaların çekirdek alanında çalışmasına rağmen, RISC-V mimarisinin sunduğu makine ve süpervizör kipleri donanım tabanlı ek bir güvenlik katmanı oluşturarak uygulamaların daha kontrollü bir ortamda yürütülmesini sağlamaktadır.
- **Statik Kod Analizi ve Sıkı Test Süreçleri:** Sistem güvenilirliğini artırmak için geliştirme sürecinde statik kod analizi araçlarının ve kapsamlı test süreçlerinin kullanılması kritik öneme sahiptir. Bu yöntem, uygulama modüllerindeki hataların erken aşamada tespit edilmesine olanak tanımaktadır.
- **Dinamik Çalışma Zamanı Ortamının Sınırlılıkları:** Çalışmada geliştirilen çekirdek içi betik dili esneklik sağlamasına rağmen, güvenlik açısından yeni bir saldırı yüzeyi ortaya çıkarmaktadır. Global yorumlayıcı kilidi mekanizması bu riski kısmen azaltmaktadır; ancak kod doğrulama, erişim kontrolü ve izolasyon teknikleri gelecekte ele alınması gereken önemli araştırma konularıdır.
- **Sanallaştırma ile İzolasyon:** PusOS'un sanallaştırma ortamlarında çalışabilmesi, kritik görevlerin farklı sanal makinelerde izole edilmesine olanak tanıyarak sistem bütünlüğünü güçlendirmektedir.
- **Hata Kurtarma ve Göç Mekanizmaları:** Deneysel sonuçlar, hata kurtarma ve uygulama göçü mekanizmalarının beklenmedik kesinti veya çökme durumlarında hizmet sürekliliğini koruduğunu göstermektedir.

Öte yandan, PusOS'un taban aldığı LittleKernel çekirdeğinin NVIDIA tarafından Güvenilir Yürütme Ortamı (GYO) kapsamında kullanılması, minimalist çekirdeklerin güvenlik odaklı ortamlarda da uygulanabilirliğini göstermektedir [34]. Bu durum, PusOS'un gelecekte donanım tabanlı GYO çözümleri ile bütünleşerek yalnızca performans değil aynı zamanda güvenilirlik açısından da güçlü bir alternatif sunabileceğini ortaya koymaktadır. Ancak donanım tabanlı GYO çözümleri düşük maliyetli cihazlarda her zaman uygulanabilir değildir. Bu nedenle, donanım desteği bulunmayan gömülü sistemler için yazılım tabanlı hafif güvenli yürütme ortamlarına yönelik güncel çalışmalar, ek

güvenlik katmanları sağlayarak önemli bir tamamlayıcı yaklaşım ortaya koymaktadır [35]. Dolayısıyla, PusOS'un ilerleyen aşamalarda hem donanım tabanlı hem de yazılım tabanlı güvenlik çözümlerini bütünleştirmesi, mevcut risklerin azaltılması açısından doğal bir araştırma yönü olarak değerlendirilebilir.

Sonuç olarak, PusOS'un çekirdek alanında çalışma prensibi performans ve verimlilik hedefleri doğrultusunda stratejik bir tercih sunmaktadır. Bununla birlikte, bu mimarinin doğası gereği ortaya çıkan güvenlik ve kararlılık risklerinin azaltılabilmesi için minimalist çekirdek tasarımı, donanım tabanlı ayrıcalık seviyeleri, sıkı yazılım geliştirme metodolojileri ve sanallaştırma tabanlı izolasyon gibi mekanizmaların bütüncül bir biçimde entegre edilmesi gerekmektedir.

7. Sonuç ve Gelecek Çalışmalar

Bu çalışma, yalnızca denetleyici modda çekirdek alanını kullanan ve özel olarak sis bilişim için tasarlanmış, RISC-V tabanlı açık kaynaklı bir işletim sistemi olan PusOS'u tanıtmaktadır. Sistemin tasarımı, uygulanması ve deneylerle doğrulanması ayrıntılı olarak sunulmuştur. Sis bilişim için kritik öneme sahip olan 10 ms'nin altındaki yanıt süreleri, veri toplama ve depolama yönetimi testleriyle başarıyla gösterilmiştir. Giriş bölümünde vurgulandığı gibi, Linux gibi kullanıcı alanına sahip işletim sistemlerinde çekirdeğe yapılan sistem çağrılarının getirdiği yüklerin maliyetleri ortaya konmuştur.

Şekil 4'te karşılaştırmalı olarak görüldüğü üzere, her ne kadar 10 ms altındaki gecikme değerlerinde Linux, PusOS'tan 3.6 kata kadar daha hızlı dosya G/Ç işlemi gerçekleştirse de 10 ms üzerindeki gecikmelerde PusOS, ortalama mesaj işleme süresinde Linux'u önemli ölçüde geçmektedir. Özellikle 50 ve 100 ms gecikme süreleri arasındaki ortalama değerlerde PusOS'un Linux rakibinden 5.5 kata kadar daha hızlı çalıştığı gözlemlenmiştir. Bu bulgular, PusOS'un belirli sis bilişim uygulamaları için sunduğu performans avantajını net bir şekilde ortaya koymaktadır.

Ayrıca, testlerimiz aracılığıyla sis bilişim için gerekli olan işletim sistemimizin hata toleransı, uygulama göçü ve sanallaştırma yetenekleri başarıyla doğrulanmıştır. İlk olarak, başarısız olan bir sis düğümünün ağa geri kazandırılması ile sistemin hata toleransı gösterilmiştir. Ayrıca PusOS üzerinde çalışan bir uygulamanın başka bir ana bilgisayara aktarılması veya göç ettirilmesi testi yapılarak etkin yük dağıtımı sağlanması doğrulanmıştır. Bu deneyler topluca PusOS'un sağlam bir işletim sistemi tasarımına sahip olduğunu ve Sis bilişim ortamları için ideal bir aday olduğunu göstermektedir.

Gelecek çalışmalar için, bu çalışmada ilk olarak sanal makine üzerinde tasarlanıp uygulanmış olan işletim sistemi, gerçek dünya platformunda doğrulanacaktır. Bu adım, sistemin gerçek yaşam senaryolarındaki uygulanabilirliğini değerlendirmek açısından kritik önem taşımaktadır. Ayrıca, derleme sistemi geliştirilecek ve sistemin kompaktlığı artırılarak, alan odaklı uygulamaların daha verimli dağıtımı sağlanacaktır.

Bu çalışma, birinci yazarın ikinci yazar danışmanlığında hazırladığı doktora tezinden üretilmiştir [36].

Kaynakça

[1] Phaphoom, N., Oza, N., Wang, X., Abrahamsson, P. *Does cloud computing deliver the promised benefits for IT industry?*, Association for Computing Machinery, New York, NY, USA, 2012, pp. 45–52.

[2] Yang, S. IoT Stream Processing and Analytics in the Fog, IEEE Communications Magazine, 2017, pp. 21–27.

[3] Misirli, J., Casalicchio, E. An Analysis of Methods and Metrics for Task Scheduling in Fog Computing, Future Internet, 2023, volume 16.

[4] Bonomi, F., Milito, R., Zhu, J., Addepalli, S. Fog computing and its role in the internet of things, Association for Computing Machinery, New York, NY, USA, 2012, pp. 13–16.

[5] Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., Jue, J. P. All one needs to know about fog computing and related edge computing paradigms: A complete survey, Journal of Systems Architecture, 2019, pp. 289–330.

[6] Al-Dulaimy, A., Jansen, M., Johansson, B., Trivedi, A., Iosup, A., v.d. The computing continuum: From IoT to the cloud, Internet of Things, 2024, volume: 27.

[7] Forti, S., Pagiaro, A., Brogi, A. Simulating FogDirector Application Management, Simulation Modelling Practice and Theory, 2020.

[8] Pop, P., Zarrin, B., Barzegaran, M., Schulte, S., Punnekkat, S., Ruh, J., Steiner, W. The FORA Fog Computing Platform for Industrial IoT, Information Systems, 2021.

[9] Cheng, B., Fuerst, J., Solmaz, G., Sanada, T. Fog Function: Serverless Fog Computing for Data Intensive IoT Services, IEEE International Conference on Services Computing (SCC), 2019, pp. 28–35.

[10] Coutinho, A., Greve, F., Prazeres, C., Cardoso, J. Fogbed: A Rapid-Prototyping Emulation Environment for Fog Computing, IEEE International Conference on Communications (ICC), 2018, pp. 1–7.

[11] Gupta, H., Dastjerdi, A. V., Ghosh, S. K., Buyya, R. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments, Software: Practice and Experience, 2017, pp. 1275–1296.

[12] https://github.com/milisarge/pusos (24.11.2025 tarihinde erişildi.)

[13] Naveed, G., Sajak, A.B., Rehan, Q., Megat, Z. A Review of Fog Computing Concept, Architecture, Application, Parameters and Challenges, JOIV: International Journal on Informatics Visualization, 2024, pp. 564-575

[14] Will, N.C., Maziero C.A. Intel Software Guard Extensions Applications: A Survey, ACM, 2023, volume 55-14.

[15] Bajer, M. Securing and Hardening Embedded Linux Devices - case study based on NXP i.MX6 Platform, International Conference on Future Internet of Things and Cloud (FiCloud), 2022, pp. 181–189.

[16] do Rosario, V. M., Pisani, F., Gomes, A. R., Borin, E. Fog-Assisted Translation: Towards Efficient Software Emulation on

Heterogeneous IoT Devices, IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2018, pp. 1268–1277.

[17] Waterman, A., Lee, Y., Patterson, D. A., Asanović, K. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0, University of California, Berkeley, 2014.

[18] Ince, M.N., Günay, M., Ledet, J. Lightweight distributed computing framework for orchestrating high performance computing and big data, Turkish Journal of Electrical Engineering and Computer Sciences, 2022, pp. 1571–1585.

[19] Choi, N., Kim, D., Lee, S. J., Yi, Y. A Fog Operating System for User-Oriented IoT Services: Challenges and Research Directions, IEEE Communications Magazine, 2017, pp. 44–51.

[20] Benomar, Z., Longo, F., Merlino, G., Puliafito, A. Enabling Container-Based Fog Computing with OpenStack, International Conference on Internet of Things (iThings), 2019, pp. 1049–1056.

[21] Barzegaran, M., Cervin, A., Pop, P. Performance Optimization of Control Applications on Fog Computing Platforms Using Scheduling and Isolation, IEEE Access, 2020, pp. 104085–104098.

[22] https://github.com/littlekernel/lk (24.11.2025 tarihinde erişildi.)

[23] Popić, S., Pezer, D., Mrazovac, B., Teslić, N. Performance evaluation of using Protocol Buffers in the Internet of Things communication, International Conference on Smart Systems and Technologies (SST), 2016, pp. 261-265.

[24] https://github.com/littlefs-project/littlefs (24.11.2025 tarihinde erişildi.)

[25] Scholz, D., Raumer, D., Emmerich, P., Kurtz, A., Lesiak, K., Carle, G. Performance Implications of Packet Filtering with Linux eBPF, International Teletraffic Congress (ITC 30), 2018, pp. 209–217.

[26] Li, J., Jin, J., Yuan, D., Zhang, H. Virtual Fog: A Virtualization Enabled Fog Computing Framework for Internet of Things, IEEE Internet of Things Journal, 2018, 5 (1), pp. 121-131.

[27] Tseng, F.H., Tsai, M.S., Tseng, C.W., Yang, Y.T., Liu, C.C., Chou, L.D. A Lightweight Autoscaling Mechanism for Fog Computing in Industrial Applications, IEEE Transactions on Industrial Informatics, 2018, 14 (10), pp. 4529-4537.

[28] https://libvirt.org (24.11.2025 tarihinde erişildi.)

[29] https://qemu-project.gitlab.io/qemu/interop/qemu-qmp-ref.html (24.11.2025 tarihinde erişildi.)

[30] Costa, B., Bachiega, J., Carvalho, L.R., Rosa, M., Araujo, A. Monitoring fog computing: A review, taxonomy and open challenges, Computer Networks, 2022, 215, pp. 109189.

[31] https://www.qemu.org (24.11.2025 tarihinde erişildi.)

[32] https://mls.akdeniz.edu.tr (24.11.2025 tarihinde erişildi.)

[33] https://github.com/littlefs-project/littlefs/issues/226 (24.11.2025 tarihinde erişildi.)

[34] https://trustedfirmware-a.readthedocs.io/en/v2.2/components/spd/tlk-dispatcher.html (24.11.2025 tarihinde erişildi.)

[35] Cecílio, J., de Sá, A. O., Jäger, G., Souto, A., Casimiro, A. LWSEE: Lightweight Secured Software-Based Execution Environment, Internet of Things, 2025, 30, 101513.

[36] https://tez.yok.gov.tr/UlusalTezMerkezi/TezGoster?key=1pwTzRXnomYf6jwqVORfUTSyXNY_ytTIs3MjWHLb6-Cn1cdjqUzXhO-nLQ-QoV0E (24.11.2025 tarihinde erişildi.)