Çoklu–Süreçli Ortamlarda Silme Kodlama Yöntemleri'nin Çoklu–Dizin Uygulaması
Implementation of Multi-threaded Erasure Coding under Multi-Processing Environments

Şuayb Ş. Arslan[1]
[1]Bilgisayar Mühendisliği Bölümü, MEF Üniversitesi, İstanbul, Türkiye
arslans@mef.edu.tr

***Özetçe* —Galois alan aritmetiği depolama ve iletişim cihazlarını veri kayıplarına karşı korumak için Reed-Solomon silme kodlarının temelini oluşturmaktadır. Galois alan aritmetiğinin en güncel uygulamaları hızlı Galois alan hesaplamaları yapmamıza imkan sağlayan Intel'in SIMD eklerinde olduğu gibi 128-bitlik işlemci vektör talimatlarına dayanmaktadır. Buna rağmen, bu uygulamalar çoklu–dizin ve çoklu–süreçli ortamlara göre optimize edilmemiştir. Diğer taraftan, sunucuların çoklu istekleri eş zamanlı olarak yerine getirmesi ve donanımın sağladığı tüm paralelliği kodlama yükünü etkili yürütmek için kullanması arzu edilmektedir. Bu makale silme kodlarının çoklu-dizin işlemcilerle çoklu–süreçli ortamlarda nasıl kullanılacağının detaylarını sunmakta ve tek dizinli uygulamalara göre emtia mikro işlemciler ve Jerasure 2.0 yazılım kütüphanesini kullanarak önemli ölçüde performans artışının olabileceğini göstermektedir.**

Anahtar Kelimeler—Silme Kodlama, Reed-Solomon, Galois alanları, Çoklu–dizin ve Çoklu–Süreçler.

***Abstract*—Galois Field arithmetic forms the basis of Reed-Solomon erasure coding techniques to protect storage or communication systems from failures. Most recent implementations of Galois Field arithmetic rely on 128-bit vector instructions, such as Intel's Streaming SIMD Extensions, which allows us to perform fast Galois Field computations. However, these implementations are not optimized for multi-threaded and multi-processing environments. Most of the real servers address multiple requests concurrently and it is desirable to use all the parallelism that hardware provides to efficiently handle the coding workload. This short paper gives some of the details about how to leverage multi-threading capabilities of the modern hardware in multi-processing environments, and demonstrates the significant performance improvements over the single-threaded applications on commodity microprocessors using Jerasure 2.0 library as a case study.**

Keywords—Erasure coding, Reed-Solomon, Galois field, multi-threading and multi-processing.

I. GİRİŞ

Silme ve düzeltme kodları bulut depolama ve haberleşme sistemlerinin veri koruma özelliklerinin altında yatan teknolojiye temel sağlamaktadır [1]. CleverSafe [2], Netapp ve Microsoft [1] gibi bir çok veri depolama şirketi artık ürünlerinde RAID-5 [3] sistemlerinin sağlayabildiğinden daha fazla kullanılabilirlik (availability) sunmak için silme kodları kullanmakta ve aktif olarak bu konuyu araştırmaktadır. En bilinen silme kodları Reed-Solomon (RS) kodlama sistemleridir [4]. Bu kodlar, verilen veri bloklarına eş veya parite (parity) bloklar oluşturarak, veriyi olduğundan daha fazla hale getirirler. Bu işlem *şifreleme* (encoding) olarak da bilinir. Şifreleme işlemi sırasında oluşturulan bu fazlalık daha sonra veri bloğu kaybolmalarında ve veri bloklarına ulaşma sorunu yaşandığında esas verinin geri kazanılmasında ya da *deşifre* (decoding) edilmesinde sisteme yardımcı olur. RS kodlama hesapları lineer Galois alan aritmetik işlemlerine dayanmaktadır. Fakat bu alanda yapılan hesaplamalar, özellikle "çarpma" işlemi ve bu işlemin büyük veri yığınları için tekrar tekrar yapılması, RS kod kullanımının yaygın olarak kullanılmasını engellemektedir. Bu işlemlerin hızlı bir şekilde yapılabilmesi için farklı yöntemler geliştirilmiştir. Bu yöntemlerin bir kısmı işlemlerin algoritmik tarafını kolaylaştırmaya çalışırken [5], diğer bir kısmı ise modern işlemcilerin veya grafik işlemcilerinin donanımsal olarak sunduğu paralelliğin etkin şekilde kullanılması üzerine yoğunlaşmaktadır [6], [7].

Silme kodlarının büyük uygulama kısmı tescilli olsa da, açık kaynak uygalamaları farklı guruplar tarafından geliştirilmiştir [8]. Bunların en güncel ve popüler olanı Jerasure 2.0 yazılım kütüphanesidir ve Intel'in ek vektör talimatlarını (SIMD) kullanmasından dolayı performansının önceki açık kaynaklı silme kodlarına göre daha yüksek olduğu gözlenmiştir [8]. Geleneksel olarak, her ne kadar bu yazılımlar vektör talimatları ile işlemsel paralellik gösterse de, temelde tek dizinli (single-threaded) olarak yapılandırılmışlardır. Bunun bir nedeni çoklu–süreçli ortamlarda yukarı katman yazılımının çoklu-dizin olarak yapılandırılmış olmasıdır. Böylelikle her dizin (thread) kendi verisi, şifreleme ve deşifreleme işlemlerinden mesûl olur. Bu aynı zamanda veri lokalizasyonu da sağlar. Diğer taraftan, bütün bu işlemler sadece şifreleme/deşifreleme işlemlerinden ibaret değildir. Özellikle veri girdi/çıktı (I/O) işlemleri ulaşılan depolama aygıtına bağlı olarak değişen zaman kayıplarına neden olabilmektedir ve bu yüzden işlemci şifreleme/deşifreleme işlemleri öncesi ve sonrasında veri okuma/yazma işlemleriyle de uğraşır. Bu ek işlemler mikroişlemciyi tam randımanlı çalıştıracak yükte değillerdir. Dahası, güncel veri depolama aygıtları işlemciye göre daha yavaş çalışan aygıtlar olduğu için işlemciler yer

Bu çalışma Quantum Corporation, TÜBİTAK 2232 ve MEF Üniversitesi tarafından desteklenmiştir.

978-1-5090-1679-2/16/$31.00 ©2016 IEEE

yer beklemeler yapabilmekte, bu da donanımın kapasitesinin altında kullanılmasına neden olmaktadır. Bunu önlemek için pek çok yöntem önerilebilir. Bu yöntemlerden bir tanesi, çoklu-süreçli ortamlarda işlemcinin bir süreç için duraksadığında diğer süreç için işlem yapmaya devam edebilmesine olanak sağlayan yazılım ve sistemlerin tasarımıdır. Biz bu çalışmamızda silme kodlarının çoklu-dizin uygalamalarının nasıl yapılacağını gösterip, bu uygulamanın örnek olarak seçtiğimiz çoklu-süreçli ortamda gösterdiği performansı sunacağız. Yaptığımız sunucu tabanlı simülasyonlara göre işlemcinin randımanın büyük oranda arttığını ve sunucu cevap süresinin kısaltmanın mümkün olabileceğini göstereceğiz.

II. Silme Kodlama, Jerasure 2.0 Kütüphanesi ve Çoklu-Dizin Uygulaması

A. Silme Kodlamaya Genel Bakış

Bir $[n, k]$ silme kodu şifrelemesi, k tane veri bloğundan $m = n - k$ tane parite bloğu oluşturur. Eğer veri bloklarını $\mathbf{d} = (d_0, d_1, \ldots, d_{k-1})$ şeklinde, ve parite bloklarını $\mathbf{c} = (c_0, c_1, \ldots, c_{m-1})$ şeklinde gösterecek olursak, aralarındaki lineer matematiksel ilişki aşağıdaki gibi yazılabilir.

$$c_i = \sum_{j=0}^{k-1} g_{i,j} d_j, \text{ bütün } 0 \leq i < m \text{ için.} \quad (1)$$

Diğer bir ifadeyle denklem (1)'i matris formunda yazıcak olursak, aşağıdaki

$$[\mathbf{d}_{1\times k} \quad \mathbf{c}_{1\times m}]^T = \mathbf{G}_{n\times k} \times \mathbf{d}^T_{k\times 1} \quad (2)$$

denkleme ulaşmış oluruz. Buradaki $\mathbf{G} = \{g_{i,j}\}$ jenaratör matrisi olarak bilinir ve sağlanması gereken lineer denklemleri tanımlar. Denklem (2)'ye dikkat edilecek olursa, jenaratör matrisi $\mathbf{G} = [\mathbf{I}_{k\times k} \quad \mathbf{P}_{k\times m}]^T$ birim matris $\mathbf{I}_{k\times k}$'yı içerir ve böylece veri, kodlama işleminden etkilenmeden şifrelenmiş olur. Bu tür kodlara *sistematik* RS kodları denmektedir. Yine denklem (2)'den görüleceği üzere parite blokları Galois alan operasyonları kullanarak matris çarpma operasyonu ile elde edilebilir.

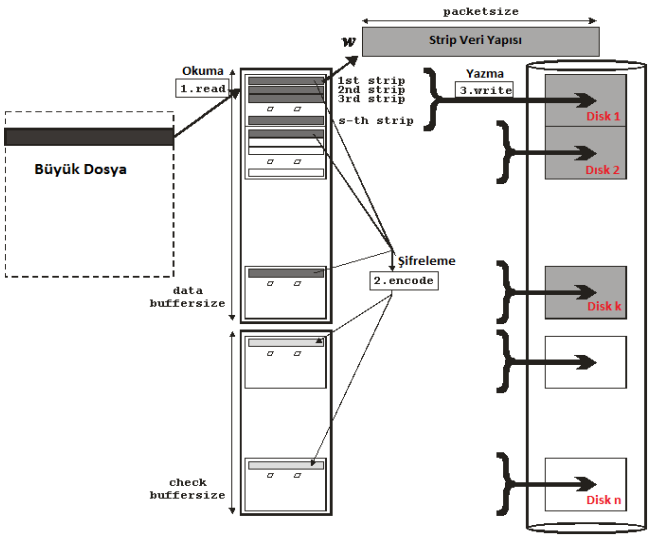
$$\mathbf{c}^T_{m\times 1} = \mathbf{P}_{m\times k} \times \mathbf{d}^T_{k\times 1} \quad (3)$$

Böylece şifreleme algoritmasını bulma işi uygun $\mathbf{P}_{m\times k}$ matrisini bulmaya dönüşmüş olur. $\mathbf{P}_{m\times k}$ matrisi girdileri w-bit kelime uzunluklu Galois alanindaki elemanlarından seçilir, yani $g_{i,j} \in GF(2^w)$. Bu girdi elemanları çeşitli şekillerde seçilebilir. Fakat $n = k+m$ bloktan herhangi k tanesinin kaybolmamış olması halinde deşifrelemenin doğru çalışabilmesi için özel seçime tabidirler. Bu özelliği taşıyan ve en bilinen jenaratör matrisleri *Vandermonde* $(g_{i,j} = (\alpha^i)^{j-1})$ [1] ve *Cauchy* matris $(g_{i,j} = 1/(x_i + y_i)$ farklı $x_i, y_i \in GF(2^w)$ için) yapılarıdır.

Deşifreleme esnasında, kullanılabilir k' veri ve m' parite bloğu olduğu varsayılırsa, şifrelenmiş n bloktan $i_1 \leq i_2 \leq \cdots \leq i_{k'}$ öncülünü sağlayan $i_1, i_2, \ldots i_{k'}$ indekslerindeki veri blokları ve $i_{k'+1} - k \ldots i_{k'+m'} - k$ indekslerindeki parite blokları kaybolmamış durumdaysa, deşifreleme algoritması bu indekslerdeki $\mathbf{G}$ matrisinin sıra vektörlerini kullanarak oluşturduğu altmatrisin $\mathbf{G}'_{k\times k}$ tersini aynı indeks sırasındaki kaybolmamış veri $\mathbf{d}' = (d_{i_1}, \ldots, d_{i_{k'}})$ ve parite $\mathbf{c}' = (c_{i_{k'+1}-k}, \ldots, c_{i_{k'+m'}-k})$ bloklarıyla ile işleme sokarak, kaybolmuş veri bloklarını bulur.

$$\mathbf{d}^T_{k\times 1} = {\mathbf{G}'_{k\times k}}^{-1} \times [\mathbf{d}' \quad \mathbf{c}']^T_{k\times 1}. \quad (4)$$

Daha sonra, kaybolmuş parite blokları bütün veri blokları kullanılarak şifreleme yoluyla bulunurlar. Veri deşifreleme işlemi için, matris tersini alma işlemi ${\mathbf{G}'_{k\times k}}^{-1}$ tek sefer yapılsa da, denklem (4)'in ifade ettiği çarpma işlemi defaatle yapılması gerekmektedir. Bütün bu işlemlerin eş zamanlı yapılabilmesi, araştırmacıları oldukça verimli algoritmalar ve uygulamalar geliştirmeye yöneltmiştir.



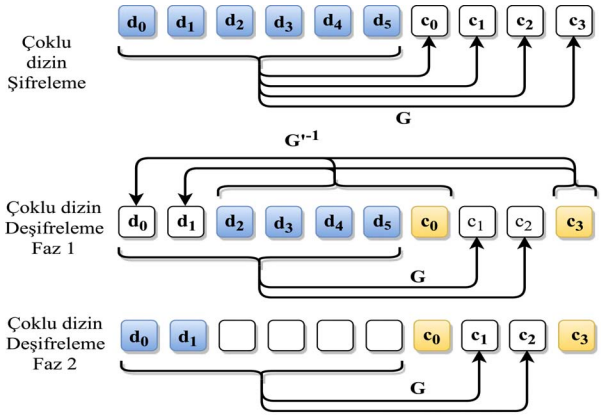
Şekil 1: Jerasure şifreleme yazılımındaki üç fazdan oluşan şifreleme metodolojisi özeti.

B. Jerasure Şifreleme/Deşifreleme Mimarisi

Her ne kadar şifreleme/deşifreleme hesaplamaları w-bitlik kelimeler üzerinde tanımlanmış olsa da, gerçek sistemlerde bayt veri kümelerinin (mesela disk sistemlerindeki sektör yapıları gibi) şifrelenmesi toplu halde Galois alanında matris toplama/çarpma işleminin birçok kez uygulanması ile gerçekleştirilir. Intel'in SIMD ekleri, bir talimatla birden fazla temel işlem yapılmasına olanak sağladığı için, bu ek talimatları kullanarak Galois alanı operasyonlarını oldukça hızlandırmanın mümkün olduğu GF-Complete [6] veya ISA-L [9] gibi yazılım kütüphaneleri ile gösterilmiştir. Jerasure şifreleme yazılımı (Vandermonde matris yapıları kullanılarak) ise GF-Complete'i kullanarak talimat düzeyinde işlemciye paralel hesap yapma olanağı vermektedir.

Jerasure şifrelemesi üç fazdan oluşmaktadır. İlk fazda, şifrelenecek olan veri yığını parçalanarak bir veri tampon belleğine yazılır. Tampon belleğin kapasitesine göre veri parçaları birden fazla sayıda olabilir. Tampon bellekteki veri k tane veri kısmından oluşur. İkinci fazın başlamasıyla bu kısımlar *strip* denilen daha küçük veri yapıları halinde işlemcinin önbellek hierarşisine göre yer değiştirir ve en son şifrelenmek için işlenir. Şifreleme hesapları sonrası oluşan parite verileri tekrar aynı hierarşiyi tersine takip ederek parite tampon belleğe kadar çıkar ve oraya yazılır. Son fazda ise, bellekteki veri kalıcı

[1] Burdaki α, yukarda ismi geçen Galois alanının primitif elemanı olarak bilinir.



Şekil 2: Jerasure çoklu–dizin şifreleme/deşifreleme uygulaması. Boş veri kutuları NULL olarak değerlendirilir.

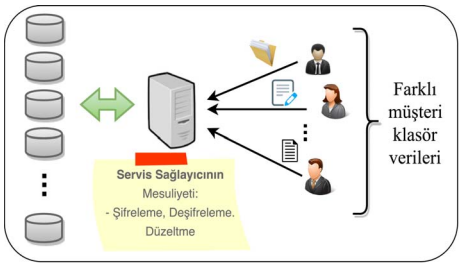
(*non-volitile*) depolama aygıtlarına yazılmak üzere gönderilir. Bu fazlar Şekil 1'de özetlenmiştir.

C. Çoklu-Dizin Uygulaması

Çoklu-Dizin uygulamaları paralellik ihitiva eden algoritmaların yeterli donanıma sahip sistemlerde performans artırılması için kullanılan metodlardan biridir. Jerasure şifreleme/deşifreleme algoritmalarının üç fazı da çoklu-dizin uygulamaları ile hızlandırılabilir. Birinci ve üçüncü fazlar genel itibari ile I/O işlemleri içermektedir. Mesela üçüncü faz *kernel* tarafında çoklu aygıtların tek bir blok aygıt olarak gösterilebilir ve "striping" (RAID0) teknikleri ile paralel hale getirilebilir [3]. Birinci fazın çoklu-dizin uygulaması ile hızlandırılabilmesi için her dizin için ayrı veri tampon belleğine ihtiyaç duyulacaktır. Bu ihtiyaç ise bellek hiyerarşisini değiştirdiği için hesaplama işlemi olarak hızlanma kaydetse bile, verinin paralel olarak tampon belleklerden önbelleklere taşınması işlemi sırasında toplam performansı olumsuz yönde etkilediği test edilmiştir. Bu bulgular bizi ikinci faza yoğunlaştırmıştır ve striplerin şifrelenmesini farklı dizinlerle hesaplama yoluna sevketmiştir. Bu yaklaşım farklı parite verilerinin veya kaybolan verilerin farklı dizinler tarafından hesaplanması şeklinde de uygulanabilir.

1) Şifreleme (Encoding): Denklemler (1) ve (3)'den anlaşılabileceği gibi, her bir parite bloğu c_i, i dizini tarafından hesaplanırsa, m tane dizin beraber çalışarak şifreleme işini paylaşabilir. Fakat burada verinin ortak önbelleklerde tutulması veri transfer işlemlerini azaltacağından, bizim bu uygulamamızda veri paylaşan dizinlerin aynı işlemci çekirdeklerinde olmasına gayret edilmiş ve böylece gereksiz veri transferleri önlenmeye çalışılmıştır. Çoklu–dizin uygulaması kabaca Şekil (2)'de $k = 6$ ve $m = 4$ için gösterilmiştir.

2) Deşifreleme (Decoding): Deşifreleme işleminin seri bölümleri şifreleme işlemine göre daha fazladır. Ayrıca çoklu-dizin uygulamasının gerçekleşebilmesi veri ve parite bloklarının ne kadarının hangi oranda kaybolmasına da bağlı olarak değişiklik göstermektedir. Mesela kaybolmuş blokların hepsi veri bloğu olduğu durumda, her bir kayıp veri bloğu $\mathbf{G'}^{-1}$ kullanılarak ayrı ayrı dizinler tarafından hesaplanabilir. Böylece maksimum sayıda dizin kullanarak deşifreleme işlemi paralel şekilde gerçekleştirilmiş olur.



Şekil 3: Veri depolama sistemleri çoklu–süreçli ortamlara verilebilecek güzel örneklerden biridir.

Tablo I: Performans ölçümlerinde kullanılan işlemcinin özellikleri.

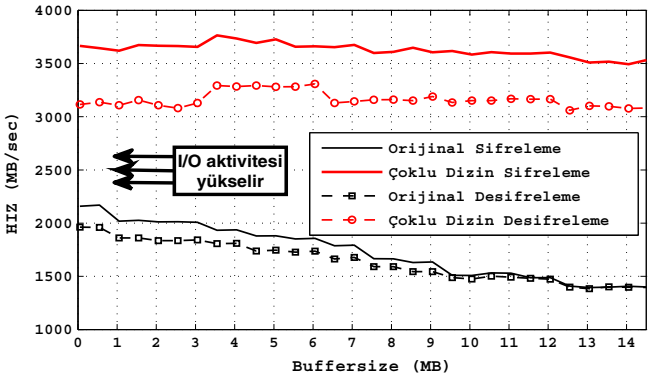
İşlemci Özelliği	Adı:Intel Xeon CPU E5620	
	Sayı/Değer	Açıklama
Soket	2	
Çekirdek	8	Her sokette 4
Dizin	16	Her çekirdekte 2
Mimari	IA64	64-bit
L1 önbellek	32KiB	
L2 önbellek	256KiB	
L3 önbellek	12288KiB	
Ana bellek	$\approx$ 49GiB	

Diğer taraftan eğer veri bloklarında $k'' = k - k'$ kayıp, parite bloklarında m'ı kayıp ($k'' + m'$ı $\leq m$) gerçekleşirse, deşifreleme yükü iki seri bölümlü iş yüküne dönüşür. Birinci işyükü k'' kayıp veri bloklarının hesaplanması ve m'ı kayıp parite bloklarının kısmi hesaplanmasından müteşekkildir. Böylece toplam $k'' + m'$ı dizin kullanarak bu işyükü paralel gerçekleştirilebilir. İkinci iş yükü ise kayıp veri bloklarının hesaplanmasından sonra bunların kullanılarak parite bloklarının diğer kısmi hesaplamaları gerçekleştirilmesi ve oluşan sonucu önceki kısmi hesaplamaya eklenmesi neticesinde parite bloklarının tamamen hesaplanmasını kapsar. Bu iki fazlı deşifreleme yöntemi, silme kodunun lineer tabanlı olmasından dolayı parite bloklarını doğru şekilde hesaplayacaktır.

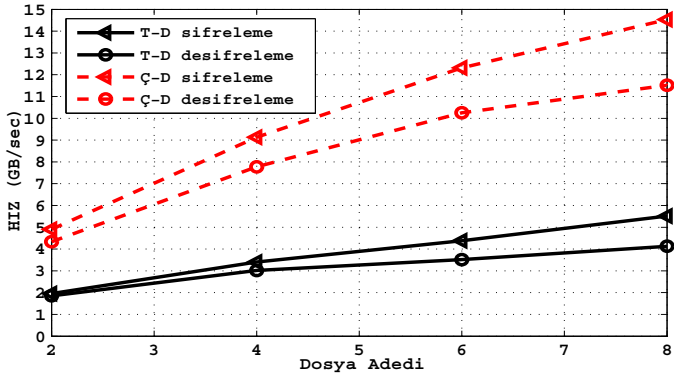
Şekil (2)'de $k'' = 2$ ve $m'' = 2$ için çoklu–dizin deşifreleme örneği iki fazlı olarak gösterilmiştir. Aynı şekilde ince karakterler kısmi hesaplama sonucunu, kalın karakterler ise nihai sonucu göstermektedir. Son olarak ifade etmek gerekir ki, ikinci fazın dizinleri, kısmi hesaplamaların sonucunu önceki fazın kısmi hesap sonuçları ile toplama işleminden de (Galois alanında XOR işlemi) sorumludurlar.

III. Çoklu–Süreçli Ortamlar ve Performans

Çoklu–süreçli ortamlar bir sunucunun birden fazla işlemci/çekirdek kullanarak farklı süreçleri çalıştırma yoluyla gelen taleplere cevap verme performansını artırma yöntemidir. Bir çok yazılım ve donanım ürünü çoklu–süreçli ortamları desteklemektedir ve böylece birçok uygulama aynı sistemde beraber çalışabilmektedir. Şekil (3) basit bir veri depolama sisteminin aynı/yakın zamanlı gelen müşteri klasör depolama taleplerini göstermektedir. Bu örnekte, sunucu gelen verileri şifreleyip arka tarafta bulunan disklere yazarak veri depolanmasını sağlamaktadır. Bunu yapan yazılım Jerasure kütüphanesini kullanarak donanımı randımanlı kullanır. Çoklu-dizin uygulaması dizinleri saf Galois alan aritmetiği yapmak için kullandığından, diğer çekirdekler eğer I/O gibi nedenlerden dolayı işkesime (inturrupt) uğradıysa, bu çekirdeklerdeki



Şekil 4: Tekli ve çoklu–dizin uygulamalarının karşılaştırmalı performansı ($k = 8$ ve $m = 4$). Sağdan sola gittikçe buffersize küçülmekte ve böylece yapılan I/O sayısı artmaktadır. Kullanılan dosya büyüklüğü = 128MiB.



Şekil 5: T–D: Tek dizinli ve Ç–D: Çoklu–dizin uygulamaların çoklu–süreçli ortamda kalasör sayısına göre karşılaştırmalı performansı ($k = 52$ ve $m = 8$).

dizinler şifreleme/deşifreleme işlemine yardımcı olur. Bu avantaj tabiki de dizinlerin donanımı kitlememesine (*mutex*) bağlıdır. Ana bellek ve önbellek kullanımının optimizasyonu ise *buffersize* ve *packetsize* parameterelerinin optimizasyonu ile sağlanmıştır. Sunacağımız sonuçlar Şekil (3)'deki sistemin simülasyonunu, üzerinde özellikleri Tablo I'da verilmiş olan işlemcili bir sunucuda test edilerek elde edilmiştir.

A. Tek Süreçli Ortamda Çoklu-Dizin Uygulama Performansı

Bu durumu bir çok k ve m değerleri için test edilmiştir. Yer kısıtlamasından dolayı, sadece $k = 8$ ve $m = 4$ için sonuçlar şekil (4)'de özetlenmiştir. Bu şekildeki x ekseni buffersize, y ekseni ise saf şifreleme/deşifreleme hızını göstermektedir; packetsize ise optimize edilmiştir. Çoklu–dizin yaklaşımı şifreleme/deşifreleme yaparken $m = 4$ dizin kullanmaktadır. Özgün tek dizinli ve çoklu–dizinli yaklaşımlar karşılaştırılırsa, performans farkı açıkça görülecektir. Ayrıca önerilen yaklaşım farklı buffersize değerleri seçilmesine rağmen daha istikrarlı sonuçlar vermektedir. Bu işlenen dosya, büyüklüğünden bağımsız yüksek performans anlamına gelmektedir. Aynı şekilde, x ekseni üzerinde sağdan sola doğru gittikçe tek dizinli yaklaşımın performansının artığı da görülür. Fakat küçük buffersize daha fazla I/O anlamına gelir ve bu durum sistemin toplam işlem süresini artıracaktır. Ayrıca sunulan sonuçların, [7]'deki çoklu–dizin uygulamasından daha iyi olduğu gözlemlenmiştir.

B. Çoklu–Süreçli Ortamda Çoklu-Dizin Uygulama Performansı

Bu durum daha yüksek işlem kabiliyeti gerektiren $k = 52$, $m = 8$, buffersize $= 5$MB için test edildi. Şekil (5)'in x ekseni toplam işlem gören dosya adedini, y ekseni ise toplam saf şifreleme/deşifreleme hızını göstermektedir. Toplam hızı en son şifreleme/deşifreleme işlemini bitiren dizin belirlemiştir. Bundan dolayı klasör sayısı 8 ile sınırlı tutulmuştur (8 çekirdekli sunucu için); çünkü daha fazlası dizinlerin birbirini beklemesini gerektirecektir. Bu hem performansı düşürür, hem de araya girecek olan I/O işlemleri zaman ölçümlerini karıştırır. Bütün simülasyon boyunca hep aynı büyüklükte (=128MiB) dosyalar kullanılmıştır. Görüldüğü üzere, çoklu–dizin yaklaşımının kalasör sayısının 2 olduğu durumda iki kat, dosya sayısı 8'e yaklaştığında ise en az üç kata kadar ilerleme gözlemlenmiştir. Bu yüksek performans son zamanlardaki GPU uygulamaları ile karşılaştırılabilecek niteliktedir [7]. GPU uygulamarına kıyasla, önerilen uygulamanın bir artısı da PCIe bandaralığı ile performansının sınırlı kalmamasıdır.

IV. Sonuç

Özellikle de yoğun trafiğin yaşandığı veri merkezlerinde, paralellik özelliği bulunan donanımların fazlalığı çoklu–dizin yaklaşımların önemini artırmış ve bu tür uygulamaların bir çok araştırmacıyı çoklu–süreçli ortamlardaki performansı araştırmaya yöneltmiştir. Bu çalışmamızda silinme kodlarının bir tip çoklu-dizin uygulamasını gösterip, çoklu–süreçli ortamlarda nasıl kullanılabiliceğini açıkladık. Tek dizinli uygulamalara göre emtia mikroişlemciler ve Jerasure 2.0 yazılım kütüphanesini kullanarak önemli ölçüde performans kazançları olduğunu nicel verilerle gösterdik.

Kaynakça

[1] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in Windows Azure storage," *In USENIX Annual Technical Conference,* Boston, June 2012.

[2] Cleversafe, Inc., "Cleversafe Dispersed Storage," Open source code distribution: http://www.cleversafe.org/downloads, 2008.

[3] Patterson, Gibson, and Katz 1988: D. A. Patterson, G. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proc. ACM SIGMOD Conf.,* Chicago, IL, June 1988, 109.

[4] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, 8:300–304, 1960.

[5] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong and S. Sankar, "Row Diagonal Parity for Double Disk Failure Correction," *FAST-2004: 3rd Usenix Conference on File and Storage Technologies*, San Francisco, CA, March, 2004.

[6] J. S. Plank, K. M. Greenan, and E. L. Miller. "Screaming fast Galois Field arithmetic using Intel SIMD instructions", *In FAST-2013: 11th Usenix Conference on File and Storage Technologies*, San Jose, February 2013.

[7] X.Chu, C.Liu, K.Ouyang, L.S.Yung, H.Liu, and Y.-W.Leung, "PErasure:A parallel cauchy Reed-Solomon coding library for GPUs", *IEEE International Conference on Communications*, London, UK, 2015.

[8] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. W. O'Hearn. "A performance evaluation and examination of open-source erasure coding libraries for storage", *In 7th USENIX FAST*, pages 253–265, 2009

[9] Intel's Intelligent Storage Acceleration Library (ISA-L). Adres: https://01.org/intel®-storage-acceleration-library-open-source-version