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 Reed-Solomon silme kodlarının genel itibariyle temelini oluşturmakta olup, depolama ve iletişim cihazlarında veri kayıplarına karşı kullanılmaktadır. Bu aritmetiğin en yeni uygulamaları 128-bitlik işlemci vektör talimatlarına bağımlı olarak geliştirilmiş, ve hızlı Galois alan aritmetiğine olanak sağlamıştır. Mesela Intel'in SIMD ek talimatları buna örnek teşkil etmektedir. Fakat, geliştirilmiş uygulamalar çoklu–dizin ve çoklu–süreçli ortamlara göre optimize edilmemiştir. Diğer taraftan, sunucular çoklu istekleri yerine getirme yetisine sahiptir ve paralel donanıma sahip olan bu sunucular kodlama yükünü daha verimli şekilde halledebilmeleri beklenmektedir. Bu kısa makale silinme kodlarının çoklu-dizin işlemcilerle çoklu–süreçli ortamlarda nasıl kullanılacağının detaylarını vermekte, ve tek dizinli uygulamalara göre emtia mikroişlemciler ve Jerasure 2.0 yazılım kütüphanesini kullanarak önemli ölçüde performans kazançlarının 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, multi-processing.

I. GİRİŞ

Silme 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 ürünlerinin içinde RAID-5 [3] sistemlerinin sağlayabildiğinin ötesinde kullanılabilirlik (availability) sunmak için silme kodları kullanmakta, ve aktif olarak bu konuyu araştırmaktadırlar. En bilinen silme düzeltme kodları Reed-Solomon (RS) kodlama sistemleridir [4]. Bu kodlar, verilen veri bloklarına eş veya parite (parity) bloklar oluşturarak veriyi olduğundan fazla hale getirir. Bu operasyon *şifrelemek* (encoding) olarak da bilinir. Şifreleme operasyonu sonucu oluşturulan bu fazlalık daha sonra veri bloğu kaybolmalarında veya veri bloklarına ulaşma sorunları yaşandığında, esas verinin geri kazanımında veya *deşifre* (decoding) edilmesinde sisteme yardımcı olur. RS kodlama hesapları lineer Galois alan aritmetik operasyonlarına dayanmaktadır. Ne varki bu alanda yapılan hesaplamalar, özellikle "çarpma" işlemi ve bu işlemin büyük veri yığınları için tekrar tekrar yapılması, RS kodlarının yaygınlaşmasını zorlaştırmıştır. Bu operasyonların hızlı şekilde yapılabilmesi için farklı metodlar geliştirilmiştir. Bu metodların bir kısmı işlemlerin algoritmik tarafını kolaylaştırmaya odaklı olurken [5], diğer bir kısmı ise modern işlemcilerin donanımsal olarak sunduğu paralelliğin etkin şekilde kullanılması üzerinedir [6].

Silme kodlarının büyük kısım uygulaması tescilli olsa da, bir çok açık kaynak uygalaması farklı guruplar tarafından çalışılmış ve araştırmacıların tasarrufuna sunulmuştur [7], [8]. Bunlarin günümüze en yakını ve popüler olanı Jerasure 2.0 yazılım kütüphanesi olup, Intel'in ek vektör talimatlarını (SIMD) kullanması dolayısı ile, performansı önceki açık kaynaklı silme kodlarına göre daha yüksek olduğu bilinmektedir [8]. Geleneksel olarak, her ne kadar bu yazılımlar vektör talimatları ile işlemsel paralellik elde etselerde, temelde tek dizinli (single-threaded) olarak yapılandırılmışlardır. Bunun bir nedeni çoklu–süreçli ortamlarda yukarı katman yazılımının zaten çoklu-dizinli olarak yapılandırılmış olmasıdır. Böylece her dizin (thread) kendi verisi, şifreleme ve deşifreleme işlemlerinden mesûl olur. Bu aynı zamanda veri lokalizasyonu da sağlar. Ne varki, bütün bu işlemler sadece şifreleme/deşifreleme hesaplamalarından ibaret değildir. Özellikle veri girdisi/çıktısı (I/O) işlemleri ulaşılan depolama aygıtına bağlı olarak değişken zaman kayıplarına neden olabilmektedir ve dolayısı ile, işlemci şifreleme/deşifreleme operasyonları öncesi ve sonrasında veri data okuma/yazma işlemleriyle de uğraşır. Bu ek işlemler mikroişlemciyi tam randımanlı çalıştıracak yükte işlemler

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

978-1-5090-1679-2/16/$31.00

değillerdir. Dahası, günümüz veri depolama aygıları genelde işlemciye göre çok yavaş çalışan aygıtlar olduğundan işlemciler yer yer beklemeler yaşayabilmektedir buda donanımın kapasitesinin altında kullanım sağlamaktadır. Bunun önüne geçmek için pek çok yöntem önerilebilir. Biz bu çalışmamızda silme kodlarının çoklu-dizin uygalamalarının nasıl yapılacağını gösterip, bu uygulamanın örnek olarak seçtiğimiz çoklu-süreçli ortamlarda nasıl performans gösterdiğini sunacağız. Yaptığımız sunucu tabanlı simülasyonlara göre, işlemcinin randımanın büyük oranda arttığını ve sunucu cevap süresinin kısaltmanın mümkün olabiliceğini 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, aşşağıdaki

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

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 matrix $\mathbf{I}_{k\times k}$'yi 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 edilebilinir.

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

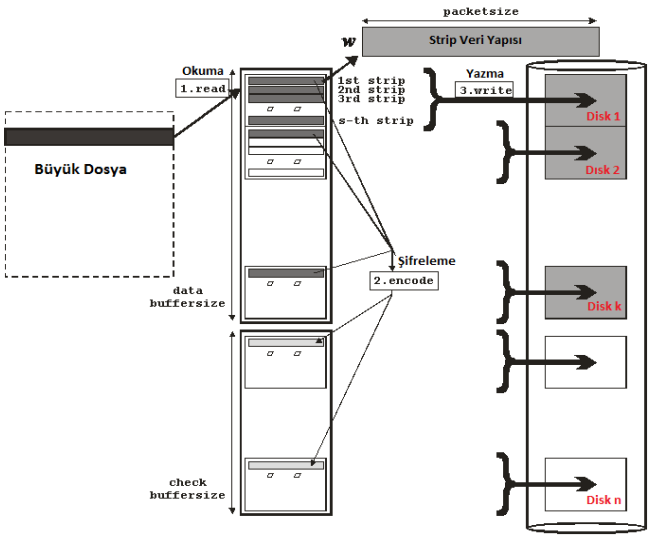
Böylece şifreleme algoritmasını bulma işi uygun $\mathbf{P}_{m\times k}$ matrisini bulmaya dönüşmüş olur. $\mathbf{P}_{m\times k}$ matrisi girdileri w-bit kelime uzunluklu Galois alanindaki elemanlarindan seçilir, yani $g_{i,j} \in GF(2^w)$. Bu girdi elemanları çeşitli şekillerde seçilebilir. Fakat, $n = k+m$ bloktan herhangi k tanesinin kaybolmamiş 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, eğer k' kullanima müsait veri bloğu olduğu varsayılırsa, şifrelenmiş n blok'tan $i_1 \leq i_2 \leq \cdots \leq i_{k'} \leq i_{k'+1} \ldots i_k$ öncülünü sağlayan herhangi $i_1, i_2, \ldots i_{k'}$ indekslerindeki veri bloklari ve $i_{k'+1} - m \ldots i_k - m$ indekslerindeki parite bloklari kaybolmamış durumdalarsa, deşifreleme algoritması bu indekslerdeki $\mathbf{G}$ matrisinin sıra vektörlerini kullanarak oluşturduğu altmatrisin $\mathbf{G}'_{k\times k}$ tersini aynı indeks sırasındaki kaybolmamiş veri $\mathbf{d}' = (d_{i_1}, \ldots, d_{i'_k})$ ve parite $\mathbf{c}' = (c_{i_{k'+1}-m}, \ldots, c_{i_k-m})$ bloklariyla ile çarparak 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 operasyonu 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 yuksek oranda paralellik ihtiva etmesi araştırmacıları oldukça verimli algoritma ve uygulamalar geliştirmeye itmiştir.

[1]Burdaki α yukardaki bağlamda geçen Galois alani'nin primitif elemani olarak bilinir.

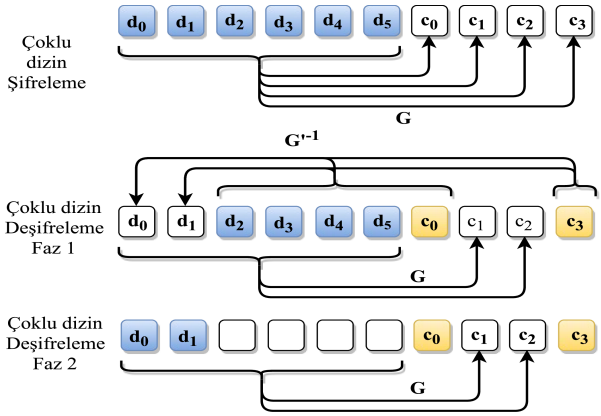


Şekil 1: Jerasure şifreleme yazılımındaki üç fazdan oluşan şifreleme metodolojisi özeti.

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 çok defa uygulanması ile gerçekleştirilir. Intel'in SIMD ekleri, bir talimatla (instruction) birden fazla Galois alan toplaması veya çıkarması yapılmasına imkan tanidigi 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) 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



Şekil 2: Jerasure çok dizinli şifreleme/deşifreleme uygulaması. Boş veri kutuları NULL olarak değerlendirilir.

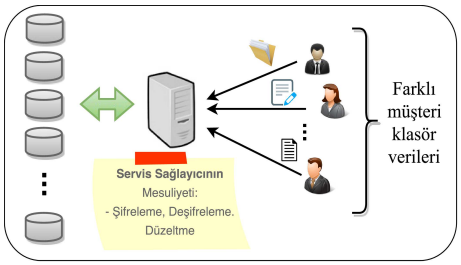
depolama aygıtlarına yazılmak üzere gönderilir. Bu fazlar özetle Şekil 1'de gösterilmiştir.

C. Çoklu-Dizin Uygulaması

Çoklu-Dizin uygulamaları paralellik ihitiva eden algoritmaların yeterli donanıma sahip sistemlerde performans artırması 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 gösterilmesi ve "striping" (RAID0) teknikleri ile paralel hale getirilebilinir [3]. Birinci faz çoklu-dizin uygulması ile hızlandırılabilmesi için her dizin için ayrı veri tampon belleğine ihtiyaç duyulacaktır. Bu ihtiyaç ise bellek hierarşisini değiştirdiği için hesaplama işlemi olarak hızlanma kaydedilebilse de, verinin paralel olarak tampon belleklerden önbelleklere taşınması işlemi toplam performansi olumsuz yönde etkilediği test edilmiştir. Bu buluntular bizi ikinci faza yoğunlaştırmış ve strip'lerin şifrelenmesini farkli dizinlerle hesaplama yoluna sevketmiştir. Bu yaklaşım farklı parite verilerinin veya kaybolan verilerin farklı dizinler tarafından hesaplanması şeklinde de uygulanabilinir.

1) Şifreleme (Encoding): Denklemler (1) ve (3)'den anlaşabiliceği gibi, her bir parite bloğu c_i dizin i 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 engellenmeye 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 gerçeklenebilmesi yine 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 $\mathbf{G}'^{-1}$ kullanilarak ayri ayri dizinler tarafından hesaplanabilir. Böylece maksimum sayida dizin kullanarak deşifreleme işlemi paralel şekilde gerçekleştirilmiş olunur.



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

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	
Bellek	≈ 49GiB	

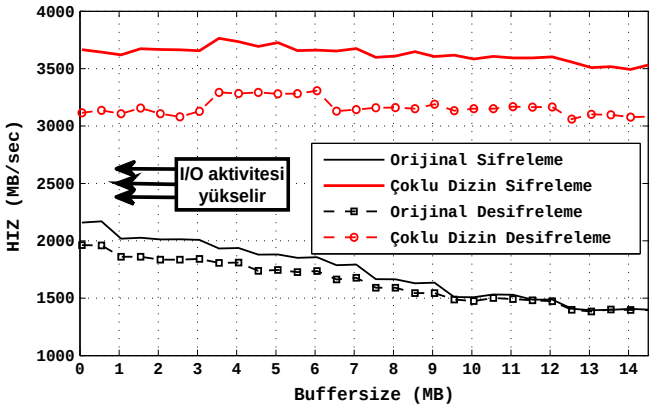
Tablo I: Performans ölçümlerinde kullanılan işlemcinin özellikleri.

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

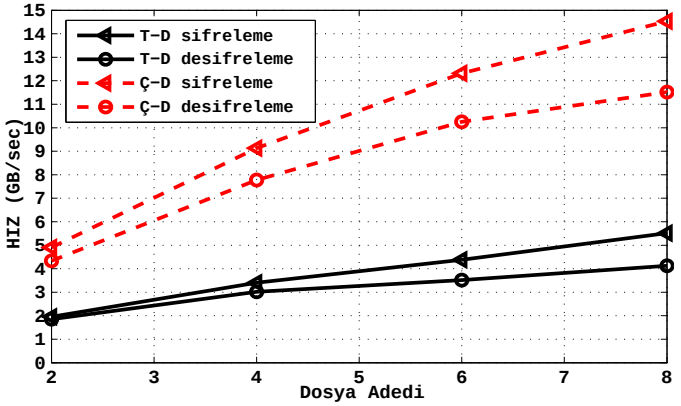
Şekil (2)'de $k'' = 2$ ve $m' = 2$ için çoklu dizin deşifreleme örneği iki fazlı olarak gösterilmiştir. Aynı şekilde ince karakterler kısmi hesaplama sonucunu, kalın karakterler ise nihai sonucu göstermektedir. Son olarak ifade etmek gerekir ki, ikinci fazın dizinleri, kısmi hesaplamaların sonucunu önceki 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 performansını artırma yöntemidir. Bir çok yazılım ve donanım ürünleri çoklu süreçli ortamları desteklemektedir ve boylece birçok uygulama aynı sistemde berber 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 depolaması 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 duraksadılarsa, bu çekirdeklerdeki dizinler şifreleme/deşifreleme



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. Kullanilan dosya büyüklüğü = 128MiB.



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

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

A. Tek Süreçli Ortamda Çoklu-Dizin Uygulama Performansı

Bu durumu bir çok k ve m değerleri için test ettik. Yer kısıtlaması dolayısı ile, 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 önerdiğimiz 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 görülür. Fakat küçük buffersize daha fazla I/O anlamına gelir, buda sistemin toplam işlem süresini artıracaktır.

B. Çoklu Süreçli Ortamda Çoklu-Dizin Uygulama Performansı

Bu durumu daha yüksek işlem kabiliyeti gerektiren $k = 52$, $m = 8$, buffersize $= 5$MB için test ettik. 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 çünkü tam o esnada bütün klasörlerin işlenmesi sona erer. Yine buna bağlı olarak klasör sayısı 8 ile sınırlı tutulmuştur (8 çekirdekli sunucu için) çünkü daha fazlası çekirdek önbelleklerinin ve dolayısı ile dizinlerin birbirini beklemesini gerektirecektir. Bu hem performansı düşürür hemde araya giricek olan I/O işlemleri zaman ölçümlerimizi karıştırır. Bütün simülasyon boyunca hep aynı büyüklükte (=128MiB) dosyalar kullanılmıştır. Sonuçlar incelendiğinde, çoklu–dizin yaklaşımı'nın kalasör sayısının 2 olduğu durumda iki kat ilerleme vaad ederken, dosya sayısı 8'e yaklaştığında bu ilerleme en az üç kata kadar çıkmaktadır.

IV. Sonuç

Özellikle de yoğun trafiğin yaşandığı veri merkezlerinde, paralellik özelliği bulunan donanımların bolluğu çok dizinli yaklaşımların önemini artırmış ve bu tür uygulamaların bir çok araştırmacıyı çoklu–süreçli ortamlardaki performansını araştırmaya yöneltmiştir. Bizde bu çalışmamızda silinme kodlarının bir tip çoklu-dizinli uygulamasını gösterip, çoklu–süreçli ortamlarda nasıl kullanılabiliceğini açıkladık ve bu yaklaşımın yüksek performans potansiyelinin olduğunu göstermeye çalıştık. Tek dizinli uygulamalara göre emtia mikroişlemciler ve Jerasure 2.0 yazılım kütüphanesini kullanarak önemli ölçüde performans kazançları olduğunu nicelleyerek 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] Z. Wilcox-O'Hearn, "Zfec 1.4.0," Open source code distribution: http://pypi.python.org/pypi/zfec, 2008.

[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%C2%AE-storage-accelerationlibrary-open-source-version