Fast and exact out-of-core and distributed K-means clustering
"Hızlı ve tam doğru, ana bellek dışında ve dağıtık K-means kümeleme"

Selçuk BAŞAK
Yıldız Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü
selcuk@selsistem.com.tr

Özet

Kümeleme (Clustering) veri madenciliğinde üzerinde en çok çalışma yapılan konulardan biridir. Kümele algoritmalarından K-means, en popüler ve yaygın kullanılanıdır. Ancak K-means algoritması veri setinin bir çok kez taranmasını gerektirmektir. Bu işlem disk üzerinde bulunan bir ana belleğe bir seferde sığmayacak kadar büyük veri setlerinde yüksek işlem maliyeti oluşturmaktadır. Yaklaşık sonuç veren K-means algoritmaları ile veri setinin taranma sayısını bir veya bir kaç keze düşürecek yöntemler mevcuttur.

Bu çalışmada hızlı ve tam doğru K-means (FEKM) algoritması anlatılacaktır. Bu algoritma ile veri setinin bir veya bir kaç sefer taranması sonucunda, K-means ile bire bir aynı sonuçlar alınabilmektedir. Bu algoritmada tüm veri setinden bir örnekleme yapılır ve bu örneklere K-means uygulanarak küme merkezleri elde edilir. Daha sonra tüm veri seti bir veya bir kaç kez taranarak bu küme merkezleri düzeltilir.

Bu çalışmada ayrıca FEKM algoritmasının dağıtık olarak çalışan şekli olan DFEKM anlatılmıştır. Bu algoritma loosely-coupled makinalar arasında dağıtılmış veri seti üzerinde işlem yapmaya uygundur ve orijinal K-means algoritması ile tam olarak aynı sonuçları verir. Bu algoritma diğer iki tam sonuç veren diğer iki algoritma olan tüm verilerin bir merkezde toplanarak K-means ve paralel K-means'ten çok daha iyi performansa sahiptir.

1 - Giriş

Clustering (Kümeleme) veri madenciliğinde üzerinde en çok çalışma yapılan konulardan biridir. Kümeleme tanım olarak, d boyutlu noktalardan oluşan bir veri setinde, f: R_1^d -R_2^d iki nokta arasıdaki mesafe veren fonksiyon olsun, birbirine benzer noktalar aynı kümede, bir birine benzemeyen noktalar ayrı kümede olmasını sağlayacak k küme merkezi hesaplanmasıdır.

İlk kümeleme teknikleri şekil tanıma ve istatistik alanlarında çok büyük sayılmayacak veri setleri üzerinde uygulanmıştır. Ancak veri madenciliği, kümeleme tekniklerinin çok büyük veri setleri üzerinde de efektif olarak uygulanmasını gerektirmektedir.

Bu çalışmada, çok büyük ve ana bellek dışında saklanan veri setleri üzerinde hızlı, veri setini bir veya bir kaç kez tarayarak, veri kümeleme işlemini sonuçları bozmadan gerçeklenmesi sağlanacaktır.

K-means kümeleme algoritması 1967 yılında MacQueen [22] tarafından geliştirilmiştir. Daha sonrasında Hartigan [16] tarafından iyileştirilmiştir. Algoritma çok sayıda pratik uygulamada yer almıştır. Orjinal K-means algoritması hafıza üzerindeki veri üzerinde çalışır ancak kolayca disk üzerindeki veriye de uygulanabilir. K-means algoritmasının temel problemi her bir turda tüm veri seti baştan sonra taranmaktadır ve yeterli kalitede bir sonuca ulaşabilmek için çok sayıda tur gerekmektedir . Algoritma, disk üzerinde bulunan büyük veri setleri ile işlem yapıldığında yüksek maliyete neden olmaktadır.

Bu problemi gidermek için bir çok çalışma yapılmış [4, 5, 10, 14] ancak bu çalışmalar orjinal K-means'e yaklaşık sonuçlar elde etmektedir. K-means'ın temel avantajı lokal minimumda sonlanmasıdır, ancak yaklaşık sonuçlarda bu elde edilemez.

Bu çalışmada K-means ile aynı sonucu veren ve veri setini daha az sayıda tarayarak bu sonuçları elde edecek bir algoritma olan FEKM algoritması geliştirildi. Bu algoritmada, öncelikle tüm veri seti hafızaya sığacak şekilde örneklenir ve bu örnek veri setine orijinal K-means uygulanır. Her turda hesaplanan küme merkezleri saklanır. Bu bilgi kullanılarak tüm veri seti bir kez taranır ve veri seti içerisindeki küme değiştirebilecek noktalar belirlenir ve buna göre küme merkezleri düzeltilir. En kötü durumda algoritma orijinal K-means kadar veri seti taraması gerektirir. Ancak, yapılan deneysel çalışmalarda algoritma en fazla 3 tarama, ortalama olarak ta 1.5 tarama ile tamamlanmaktadır. Sonuçta orijinal K-means'e göre veri tarama sayısını 2 ila 4.5 kat arasında azaltmıştır.

Bu çalışmada ayrıca FEKM algoritmasının dağıtık olarak çalışan şekli olan DFEKM algoritması anlatılmıştır. Bu algoritma loosely-coupled sistemler arasında dağıtılmış veri seti üzerinde işlem yapmaya uygundur. Bu konuda daha önce yapılmış çalışmalardan farklı olarak orijinal K-means algoritması ile tam olarak aynı sonuçları verir.

Yaptığımız deneysel çalışmalar sonucunda bu algoritma diğer iki tam sonuç veren diğer iki algoritma olan tüm verilerin bir merkezde toplanarak K-means ve paralel K-

means'ten çok daha iyi performansa sahiptir. DFEKM eşit olmayan yük dağılımı varken, tightly coupled ortamlarda bile paralel K-means'ten daha yüksek performans göstermiştir.

2 - İlgili Çalışmalar

Kümeleme algoritmaları ile ilgili literatürde oldukça geniş çalışma var. Bu konudaki ankete ilgili kitaptan [17] ve çalışmalardan [2, 13] ulaşabilirsiniz. Bu çalışmada, K-means'in iyileştirilmesi üzerine olan çalışmaları göz önüne alacağız. Moore ve Pelleg [26] , *k*-means'in bir türevinde uzaklık bilgisini saklamak için $k - d$ tree-based veri yapısını kullanıp, her turu oldukça hızlandırabildi. Bu algoritma tümü hafızada bulunan veri setlerine odaklanılmıştır. Bradley ve Fayyad [4] tek turda yaklaşık *k*-means geliştirdi. Bu algoritmda giriş noktalarını farklı merkezlere dahil olma ihtimallerine göre toplanıyor. Farnstorm [11] bu fikri daha da geliştirdi. Bu ve benzeri bir çok çalışmada tur sayısı azaltılarak yapılmıştır, ancak sonuçlar orijinal K-means'e yaklaşık elde edilmiştir.

3 - Algoritma Detayları

3.1 Ana Fikir

FEKM algoritmasının temel fikri, örnekleme ile elde edilen yaklaşık küme merkezleri veri setinin bir veya bir kaç kez taranması ile düzeltilerek orijinal K-means ile elde edilecek merkez noktalarını tam doğru olarak elde edilebileceğidir.

Bunu gerçekleştirebilmek için bazı sorunlarıncevaplamak gerekir. Örnekleme ile elde edilen yaklaşık küme merkezleri hesaplanırken hangi bilgilerin saklanması gerekir? Bu bilgi tüm veri setinin çok defa taranmasını önlemek için nasıl kullanılabilir? Orijinal K-means ile aynı merkez noktalarını hesapladığımızdan nasıl emin olacağız? Öncelikle, K-Means örneklenen veri seti üzeride tüm veri setine uygulanacak K-means ile aynı sonlanma ve ilk merkez noktaları ile çalıştırılır. Bu örnekler üzerinde K-means işlemi yapılırken her tur sonunda elde edilen k küme merkezleri saklanır. Buna ek olarak her bir küme merkez için güvenilir yarıçap (confidence radius) hesaplanır. Bu bilgi satırları turları, sütunları merkezleri gösteren, her bir gözünde bir merkez noktası ve güvenilir çap saklanan bir tabloda tutulabilir. Sonrasında, tüm veri seti bir kez taranır. Bu taramanın veri setindeki her nokta için elde edilen her bir satırı için hangi kümeye dahil olduğu bulunur. Sonrasında bu noktanın eğer tüm veri seti üzerinde K-means uygulansaydı başka bir kümeye dahil olup olmayacağı tahmin edilmeye çalışılır. Burada hedefimiz, turların her hangi birinde farklı kümeye dahil olabilecek noktaları belirleyip saklamaktır. Bu noktalar sınır noktalar olarak adlandırılır. Eğer bu noktalar belirlenir ve hafızada saklanabilirler ise veri setini tekrar taramaya gerek kalmadan sonuca ulaşabiliriz. Ancak, Bu noktalar sadece tahmin edilebilir, yani eğer tahmin hatalı ise ek veri setini tekrar baştan sona taramak gerekebilir.

Veri setindeki bir noktanın, tablodaki bilgiler ile kenar noktası olup olmadığı belirlenebilir. Eğer kenar noktası ise bir buffer'a atılır değilse ilgili kümeye ait toplam ve adet bilgisi güncellenir. Tüm veri setindeki tüm noktalar için bu işlemi yaptıktan sonra, tablodaki ilk satırdan itibaren küme merkezlerini kenar noktaları ve kümeye ait toplam ve adetlerden yararlanarak yeniden hesaplarız. Eğer yeni hesaplanan merkez noktaları güvenilir yarıçapın dışında kalırsa, kenar noktalarımızın hatalı tahmin edildiğini anlarız ve yeni merkez noktalarını başlangıç noktası alarak yeniden veri setini baştan sona tararız ve yukarıdaki adımları baştan tekrarlarız. Ancak tüm hesaplanan küme merkezleri güvenilir yarıçap içinde ise gelecek tur için bu merkezler kullanılarak devam edilir. Algoritma orijinal k-means ile aynı sonlanma koşulu ile tamamlanır.

3.2 Matematiksel İfade

Tüm veri setine orijinal K-means algoritması uyguladığımızda i. turda elde edilen k küme merkezi sırasıyla $c_{i1}, c_{i2}, \ldots, c_{ik}$ olsun ve k-means merkezi olarak adlandırılsın. FEKM ile aynı başlangıç noktaları ile i. adımda elde edilen k küme merkezi $s_{i1}, s_{i2}, \ldots, s_{ik}$ olsun ve örnek merkezleri olarak adlandırılsın. Her bir örnek merkezi s_{ij} için FEKM ile güvenilir yarıçap δ_{ij} ilişkilendirilir. İdeal olarak güvenilir yarıçap δ_{ij} küçük olmalı, ancak $d(c_{ij}, s_{ij}) \leq \delta_{ij}$ koşulunu da sağlamalıdır.

FEKM de tüm veri seti tarandığında, her bir noktanın sahibi olarak tabloda o tura ait satırdaki örnek merkezlerinden en yakın olanıdır.

Tanım 1: Veri setindeki her hangi bir p noktası için, örnek merkezlerine göre i. turdaki sahibi, s_{ij} varsayalım. $l, l \neq j$,için

$$0 \leq d(s_l^i, p) - d(s_j^i, p) \leq \delta_j^i + \delta_l^i,$$

koşulunu sağlıyorsa p noktası i. tur için kenar noktadır.

Kenar noktaların sahibi olan örnek merkezleri ile k-means merkezleri yüksek olasılıkla farklıdır. *i.*tur için veri setindeki dengeli noktalar, aynı tur için kenar nokta olarak belirlenmeyen noktalardır.

Dengeli noktalar için genelde sahibi olan örnek merkezi ile arasındaki mesafe göre diğer örnekleme merkezlerine olan uzaklığı çok daha fazladır ve matematiksel olarak, i. turdaki örnekleme merkezi s_{ij} olan bir p dengeli noktası için, Herhangi bir $l, l \neq j$ için,

$$d(s_l^i, p) - d(s_j^i, p) > \delta_j^i + \delta_l^i.$$

koşulu sağlanır.

```
Input: D_i (Data Points), S_i (Sample Data Points), InitCtrs (Initial Centres),
       ϵ (Stopping criteria in k-means algorithm)
Output: k Cluster Centres
begin
    flag ⟵ 1 ;
    while flag do
        List Buffer ⟵ NULL ;
        Index[] ⟵ NULL ;
        Table CAtable ⟵ NULL;
        NumRow ⟵ BuildCATable(InitCtrs, S_i, CAtable, ϵ) ;
        for each D_i do
            for eachrow_j ∈ NumRow do
                if (ClosestCentre⟵IsBndrPoint(D_i))
                then
                    BufferInsert(D_i) ;
                    Index[BndCnt][j] ⟵ 1 ;
                    BndCnt++ ;
                end
                else
                    UpdateSufficientStats(ClosestCtr, CAtable, D_i,
                    row_j) ;
                end
            end
        end
        for eachrow_j ∈ NumRow do
            NewCtrs ⟵ RecomputeCtrs(CAtable, Buffer, Index, row_j) ;
            if (IsCtrsWithinRadii(NewCtrs, CAtable, row_{j+1})
            then
                UpdateCATableCtrs(NewCtrs, CAtable, row_{j+1}) ;
                flag ⟵ 0 ;
            end
            else
                InitCtrs ⟵ NewCtrs;
                flag ⟵ 1;
            end
        end
    end
    OutputCATableCtrs(CAtable, row_NumRow) ;
end
```

Fig1: FEKM pseudo kodu.

3.3 Algoritmanın Açıklaması

FEKM algoritmasının pseudo kodu fig.1'de gösterilmektedir. FEKM'de kullanılan temel veri yapısı, örneklenmiş ver setinin küme merkezlerin tutulduğu tablodur. Bu tablo *küme soyut (CAtable)* tablosu olarak adlandırılır.

Algoritma CAtable'ı tüm veri setinden örneklenmiş veri setinden oluşturmakla başlar. CATable'ın her bir satırında o satıra karşılık gelen turda o küme için hesaplanan merkez ve güvenilir yarıçap bilgisini saklar. Bu tablo BuildCATable fonksiyonu ile oluşturulur. Sonrasında tüm veri seti baştan sona taranır ve tablonun her bir satırı için kenar nokta olması beklenenler belirlenir. *IsBndrPoint* fonksiyonu her bir nokta için kenar nokta olma durumunu kontrol eder.Eğer bir nokta, tablonun bir satırı için kenar nokta olarak bulunduysa, sonraki tablonun satırları için de kenar nokta olması olasıdır. Bu nedenle iki liste oluşturulur, birinde noktalar saklanır diğerinde, indekste , ise noktanın ilk listedeki yeri ve noktaların geçtiği satır numarası saklanır. Indeks listesinde noktanın geçtiği satır, noktanın geçtiği satıra karşılık gelen bit bir olarak işaretlenen bir binary değişken tutulur. Ayrıca tablonun her bir gözünde, o bir küme merkezi için kenar olmayan (dengeli) noktaların sayı ve toplamları da saklanır. *UpdateSufficientStats* fonksiyonu bunu sağlar. Sonrasında tablodaki merkez noktalarını ilgili kenar noktaları ve kenar olmayan noktaların sayı ve toplamlarından yararlanarak yeniden hesaplanır. Bu işlem *RecomputeCtrs* fonksiyonu ile yapılır. Sonrasında yapmış olduğumuz kenar nokta varsayımımızın doğruluğu yeni hesaplanan merkezler ile önceki merkezlerin güvenilir yarı çapı içinde olup olmadığına bakılarak kontrol edilir. Eğer yeni hesaplanan merkez noktalar güvenilir çember içinde ise tabloda yeni merkezler *Update-CAtableCtrs* fonksiyonu ile güncellenir. Eğer herhangi bir yeni hesaplanan merkez, güvenilir yarıçap dışında ise yeni hesaplanan merkezler ilk merkez olarak alınır ve algoritma CAtable tablosunun oluşturulmasından itibaren tekrarlanır.

3.4 Güvenilir Yarıçapın Hesaplanması

Güvenilir yarı çapın hesaplanması algoritmanın çalışmasında önemli rol oynar. Çok büyük yarıçap alınması çok fazla kenar nokta oluşmasına neden olacaktır, ve bu noktalar hafızaya sığmayabilir. Aynı zamanda çok küçük güvenilir yarıçapta, örnek küme merkezleri ile K-means küme merkezlerinin arasındaki mesafeden küçük kalıp,veri seti üzerinde birden çok taramaya neden olabilir. Bu uygulamada güvenilir yarıçap aşağıdaki yöntemle hesaplanmıştır.

$$\delta_j^i = f \times \left(\frac{\sum_{p=1}^{N} (X_p - X_c)}{N} \right)^{1/2}$$

Burada δ_{ij} yarıçapı, örneklenmiş veri seti üzerinde K-means'in i.turundaki j. küme merkezi için hesaplanmıştır. Burada Xp, j. kümeye ait olan d boyutlu noktaları ifade eder.Xc ise, j. küme merkezidir.N j. kümeye ait nokta sayısıdır. f ise deneysel olarak seçilmiştir ve uygulamada 0.05 olarak alınmıştır. Eğer δ_{ij} büyük hesaplanırsa, kenar noktalar çok çıkar ve bu noktalar hafızaya sığmayabilir. Bu durumda f değeri küçültülerek yarıçap küçültülür ve işlem hafızada yapılabilir.

4 - Performans Analizi

Bu bölümde FEKM algoritmasının performansı ile orijinal K-means çalışma süreleri karşılaştırılacaktır. Orijinal K-means algoritmasının tamamlanması için gerek tur sayısı n olsun. Tüm veri setinin baştan sona bir kere okunması için geren I/O maliyeti C_I olsun, bunun yanında her tur için tüm veri seti üzerinde yapılan hesaplama maliyeti C_c olsun.

Böylece orijinal K-means algoritması için gereken süre aşağıdaki gibi hesaplanabilir.

$T_{KM} = n \times (C_I + C_c)$.

FEKM algoritması ile veri setinin örneklenmesi sayısı P olsun. Örnekleme oranı SS olsun. Örneklemiş ver seti üzerinde K-means tur sayısı ortalama olarak m olsun.

Bunlara göre FEKM algoritması için geçen süre aşağıda belirtilmiştir.

$T_{FEKM} = P \times (SS \times (C_I + C_c) + C_I + m \times C_c)$

$= P \times (SS + 1) \times C_I + P \times (SS + m) \times C_c$.

Yukarıdaki ifadeden çoğu zaman FEKM için , orijinal K-means'e göre daha fazla hesaplama maliyeti olduğu görülür. Ancak disk tabanlı veri setlerinde işlem yaparken diskten okuma maliyeti hesaplama maliyetine göre çok daha fazladır. Ayrıca I/O işlemleri ile hesaplama işlemleri overlap ile paralel olarak yapılabilir.
Her durumda , eğer P küçükse FEKM , orijinal K-means algoritmasına göre çok daha hızlıdır. Denemeler sonucunda P, çoğu durumda 1, en fazla ise 2 veya 3 olmuştur. Ayrıca %5 veya % 10 örnekleme genelde yeterli olmuştur.

5 FEKM 'in Deneysel Sonuçları

Bu bölümde FEKM algoritmasının sentetik ve gerçek veriler üzerinde uygulanmasının sonuçlarını değerlendireceğiz. Değerlendirme orijinal K-means algoritmasının çalışma süresi ile karşılaştırarak yapılmıştır.
Buna ek olarak FEKM ile tüm veri setinin kaç defa tarandığı da incelenmiştir.
Denemeler 1-GB hafızalı 700-MHz Pentium makineler üzerinde yapıldı. Tüm denemelerde başlangıç küme merkezleri ve sonlanma koşulları her iki algoritma için aynı tutuldu. K-means algoritmasının performansı ilk küme merkezlerinin seçiminden çok etkilendiğinden, denemelerde hem iyi hem de kötü ilk merkez seçimleri kullanıldı. Ayrıca iki farklı sonlanma koşulu kullanıldı, bunlardan ilki merkezler artık çok yer değiştirmediğinde, ikincisi ise maksimum sayıda tur olarak alındı.

Table 1 Explanation of the notations used in the result tables

c*x* d*y*	Dataset with *x* clusters and *y* dimensions
I_{KM}	No. of iterations in *k*-means
Init "g"	Good initialisation
Init "b"	Bad initialisation
T_{KM}	Running time of *k*-means (s)
T_{FEKM}	Running time of *FEKM* (s)
SS	Sample size (%)
P	Number of passes by *FEKM*
se	Squared error between final centres and the centres after sampling

Tablo 1 'de değerlendirmede kullanılan kısaltmalar açıklanmıştır.

5.1 Sentetik Veri Setleri Üzerindeki Sonuçlar

Sentetik veri setleri üzerindeki değerlendirmeler 18 1.1GB ve 2 4.4 GB ver seti üzerinde yapıldı. Bu veri setleri farklı sayıda küme ve boyut içeriyordu. Sentetik veri setleri Gaussian dağılımı fonksiyonu ve random ağırlık değerleri ile elde edildi.
Sonuçlar aşağıdaki tablolarda belirtilmiştir.

Table 2 Performance of *k*-means and *FEKM* algorithms on synthetic datasets, 5 clusters

Data	I_{KM}	Init	T_{KM}	T_{FEKM}	SS	*P*
c5d200	3	g	1452.83	644.66	10	1
c5d200	3	g	1452.83	571.22	5	1
c5d100	6	b	2688.65	902.81	10	1
c5d100	6	b	2688.65	762.47	5	1
c5d50	8	b	3602.31	1114.62	10	2
c5d50	8	b	3602.31	987.43	5	2
c5d20	8	b	3313.84	1098.41	10	1
c5d20	8	b	3313.84	940.47	5	1
c5d20	2	g	829.29	507.94	10	1
c5d20	2	g	829.29	412.53	5	1
c5d10	8	g	3833.44	1633.39	10	1
c5d10	8	g	3833.44	1302.52	5	1
c5d5	6	b	3116.89	1387.13	10	1
c5d5	6	b	3116.89	1236.51	5	1

Table 3 Performance of *k*-means and *FEKM* algorithms on synthetic datasets, 10 clusters

Data	I_{KM}	Init	T_{KM}	T_{FEKM}	SS	*P*
c10d200	10	b	4808.48	1559.70	10	2
c10d200	10	b	4808.48	1324.38	5	2
c10d200	2	g	954.33	577.22	10	1
c10d200	2	g	954.33	459.52	5	1
c10d100	3	g	1081.45	435.25	10	1
c10d100	3	g	1081.45	386.49	5	1
c10d50	100	b	49144.98	11267.28	10	2
c10d50	3	g	1462.43	725.22	10	1
c10d50	3	g	1462.43	649.60	5	1
c10d20	10	b	4570.63	1708.57	10	2
c10d20	10	b	4570.63	1408.57	5	2
c10d10	3	g	1623.10	867.50	10	1
c10d10	3	g	1623.10	773.64	5	1
c10d5	100	b	60310.89	26491.76	10	2
c10d5	100	b	60310.89	19349.28	5	2

Table 4 Performance of *k*-means and *FEKM* algorithms on synthetic datasets, 20 clusters

Data	I_{KM}	Init	T_{KM}	T_{FEKM}	SS	*P*
c20d200	100	b	54862.33	27388.85	10	2
c20d200	3	g	1898.65	746.51	10	1
c20d200	3	g	1898.65	584.88	5	1
c20d100	100	b	41029.15	18106.51	10	2
c20d100	3	g	1233.12	646.75	10	1
c20d100	3	g	1233.12	585.63	5	1
c20d50	3	g	1796.30	938.90	10	1
c20d50	3	g	1796.30	882.36	5	1
c20d20	10	b	5335.15	2528.11	10	2
c20d20	10	b	5335.15	2112.42	5	2
c20d10	6	g	3919.08	1814.73	10	1
c20d10	6	g	3919.08	1643.75	5	1
c20d5	6	b	4619.95	2899.76	10	1
c20d5	6	b	4619.95	2353.41	5	1

Tablolardan görüldüğü gibi FEKM ile tüm veri seti en fazla 2 defa taranmıştır. FEKM'ın hızlandırması kötü ilk merkez değerleri ile daha yüksek olduğu görünüyor. Örnekleme %5 ile yapıldığında her zaman daha kısa sürdü, çünkü %5 ve %10 yapıldığında hep aynı tarama sayısı ile işlem sonuçlandı. Küme ve boyut sayısının değişmesinin performans üzerinde büyük bir etkisi görülmedi.

Table 5 Performance of *k*-means and *FEKM* algorithms with 4.4-GB synthetic datasets

Data	I_{KM}	Init	T_{KM}	T_{FEKM}	SS	*P*
c20d100	2	g	4393.02	2931.53	10	1
c20d100	2	g	4393.02	2204.42	5	1
c20d100	10	b	21985.62	8194.07	10	1
c20d100	10	b	21985.62	7467.53	5	1
c5d20	10	b	43254.34	10341.42	10	2
c5d20	10	b	43254.34	9632.71	5	2

Tablo 5'te 4.4 GB veri seti üzerindeki FEKM 'nin orijinal K-means'e göre 2 ila 4.5 kat hızlı olduğu görülüyor.

5.2 Gerçek Veri Setleri Üzerindeki Sonuçlar

FEKM ayrıca KDDCup99, Corel image database ve the Reuters-21578 gerçek veri setleri ile değerlendirildi. Bu veri setlerinden standart yöntemlerle elde edilen özellik vektörleri, veri setini hafızaya sığmayacak kadar büyük yapabilmek için random sampling ile çoğaltıldı.

Table 6 Performance of *k*-means and *FEKM* algorithms, real datasets

Data	I_{KM}	Init	T_{KM}	T_{FEKM}	SS	P	se
kdd99	19	g	7151	2317	10	2	4.0
kdd99	19	g	7151	2529	15	2	3.5
kdd99	19	g	7151	2136	5	2	4.2
Corel	43	g	28442	10503	10	3	2.2
Corel	43	g	28442	12603	15	3	2.15
Corel	43	g	28442	9342	5	3	3.24
Reuter	20	b	41290	10311	10	2	10.1
Reuter	20	b	41290	11204	15	2	8.6
Reuter	20	b	41290	9214	5	2	14.9

6 - Dağıtık FEKM

Bu bölümde FEKM'nin dağıtık olarak çalışan şekli DFEKM anlatılacaktır.

6.1 Motivasyon

Internet, web ve grid computing teknolojileri ilerledikçe coğrafi olarak dağıtık ortamda bulunan veri üzerinde analiz yapmak bir ihtiyaç ve problem haline gelmiştir. Veri madenciliği algoritmalarının dağıtık veri setlerine uygulanmasının zorluğu net olarak görülmüştür. Tüm veri setinin bir noktada toplanması ise çoğu zaman feasible olmamaktadır. Bunlara iletişim in gecikmelerinden dolayı paralel algoritmaların uygulanmak pek mümkün olmuyor.başarılı sonuçlar oluşturmuyor. K-means algoritması için paralel olarak çalışacak bir çok çözüm getirilmiştir ve bu algoritmalar tightly coupled paralel sistemler üzerinde başarılı olmuşlardır. Ancak dağıtık ve dengesiz dağılımlı veri setleri üzerinde kötü performans almaktadırlar.

Bu alanda yapılmış diğer çalışmalardan farklı olarak DDEKM ile K-means ile aynı sonuçlar elde edilmektedir.

6.2 Algorithma Açıklaması

Veri setinin iki veya daha fazla düğümde toplansın ve bu düğümleri veri kaynağı olarak adlandıralım. Buna ek olarak sonuçların elde edileceği düğüm, merkez düğüm olsun Veri seti veri kaynaklarına yatay olarak paylaştırılmış olsun. Dağıtık FEKM algoritmasına göre,
her bir veri kaynağından örnek alınır ve merkezi düğüme gönderir. Merkezi düğüm örneklenmiş veri seti üzerinde K-means çalıştırır. FEKM'nin ana veri yapısı olan CATable oluşturulur ve saklanır. Sonra CATable, tüm ver kaynaklarına gönderilir. Sonrasında bu tablo ile , tüm veri kaynaklarında bir kere tarama yapılır ve kenar noktaları belirlenir ve saklanır, kenar olmayan noktaların sayılır ve değerleri toplanır ve merkezi düğüme gönderir. Sonra merkezi düğümde, tablonun ilk satırından itibaren yeni merkez noktaları hesaplanır ve bu noktaların güvenilir yarı çap içinde olup olmadığına bakılır. Eğer herhangi biri değilse yeni hesaplanan merkezler başlangıç merkezi olarak alınır ve algoritmaya baştan başlanır, yoksa tüm satırlar üzerinde işlem yapılarak işlem sonlandırılır.

6.3 Deneysel Sonuçlar

Yapılan çalışmalarda elde edilen deneysel sonuçlar aşağıdaki tablolarda sunulmuştur. Deneysel çalışmalarda iki yaklaşımla karşılaştırma yapıldı, biri tüm veri setlerinin tek bir düğüme toplanarak bu düğüm üzerinde işlem yapılması, diğeri ise paralel k-means.Diğer algoritmaların sonuçları k-means'le tam olarak aynı olmadığından incelemeye alınmadı.

Table 7 Statistics from parallel *k*-means and *DFEKM* executions on synthetic and real data sets

Dataset	No. of points (millions)	Size (GB)	No. of iterations (*k*-means)	No. of passes (DFEKM)
c5d100	2	1.1	20	2
c5d10	20	1.1	20	2
c20d100	2	1.1	20	2
c20d10	20	1.1	20	2
kdd	1.5	1.8	18	2
Corel	1.5	1.9	20	2

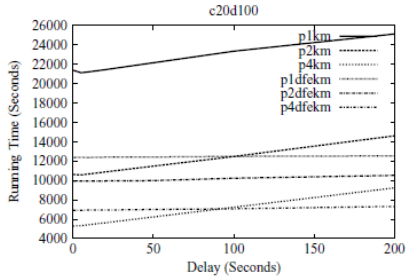


Fig. 1 Running time of *DFEKM* and parallel *k*-means with increasing delays: 1, 2 and 4 nodes dataset `c20d100`

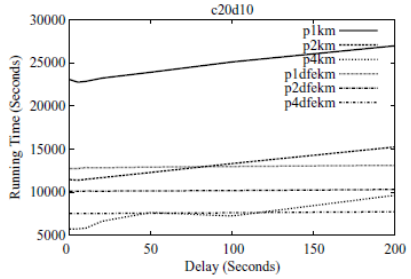


Fig. 2 Running time of *DFEKM* and parallel *k*-means with increasing delays: 1, 2 and 4 nodes dataset `c20d10`

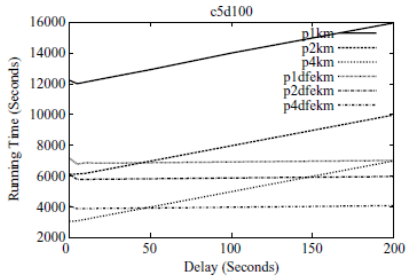


Fig. 3 Running time of *DFEKM* and parallel *k*-means with increasing delays: 1, 2 and 4 nodes dataset `c5d100`

Deneysel çalışmalardan bir çok sonuç çıkarılabilir. İlk olarak tek düğümde, DFEKM *k*-means'ten daha iyi bir performans gösteriyor. Bunun nedeni algoritmanın daha az veri seti taraması yapmasındandır. 2 ve 4 düğümde, iletişim gecikmeleri büyüdükçe *DFEKM,* paralel *k*-means'ten daha hızlı sonuçlar vermektedir. Ayrıca Fig.1-4 gösteriyor ki, tüm verinin tek bir düğüme toplanıp, tek düğümde K-means veya DFEKM yapıldığında veri transfer süresi ihmal edilse bile 2 ve 4 düğüm DFEKM daha iyi performansa sahiptir.

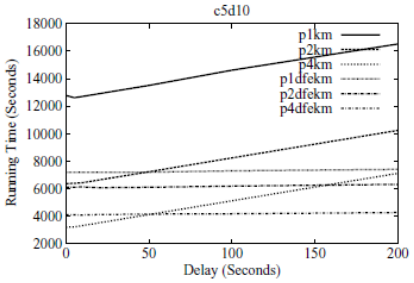


Fig. 4 Running time of *DFEKM* and parallel *k*-means with increasing delays: 1, 2 and 4 nodes dataset `c5d10`

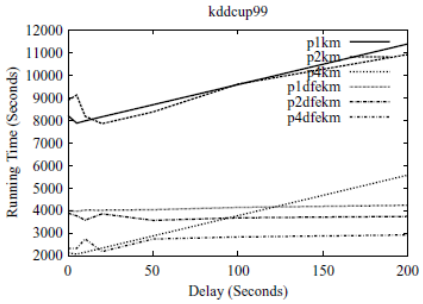


Fig. 5 Running time of *DFEKM* and parallel *k*-means in the presence of load imbalance: `c20d100` dataset

Fig.5'te üç farklı dengesiz yük dağılımı ile değerlendirme yapılmıştır . İlki 40%, 20%, 20% ve 20%, ikincisi 30%, 30%, 30% ve 10%, üçüncüsü ise 50%,20%, 20% ve 10%' dir.

İlk ve üçüncü durumda DFEKM daha iyi performans göstermiştir, bunun nedeni paralel K-means'in performansı en yavaş düğüme bağlıdır. Buradan anlaşılacağı gibi eğer yük dağılımı çok dengesiz ise DFEKM, tightly coupled sistemlerde bile daha paralel K-means'ten daha iyi performans gösterir. Aynı değerlendirmeleri gerçek veri setleri olan `KDDCup99` ve `Corel` image databases içinde yapıldığında da benzer sonuçlar elde edildi.

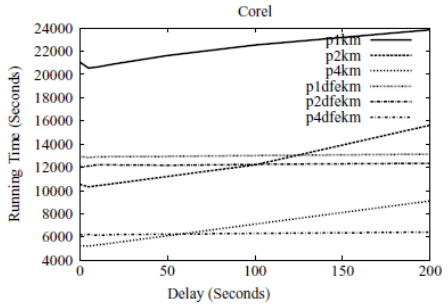


Fig. 6 Running time of *DFEKM* and parallel *k*-means with increasing delays: 1, 2 and 4 nodes `Corel` dataset

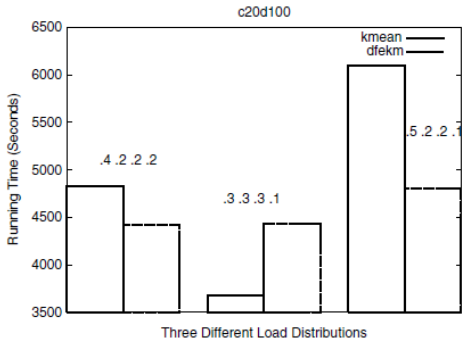


Fig. 7 Running time of *DFEKM* and parallel *k*-means with increasing delays: 1, 2 and 4 nodes `kddcup99` dataset

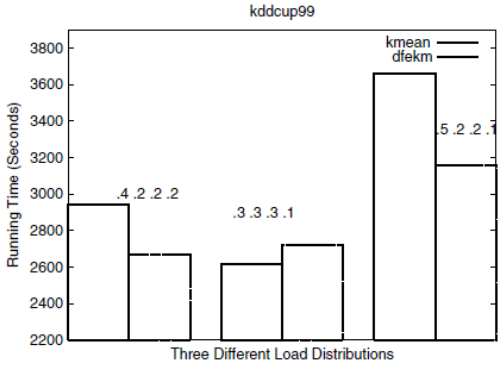


Fig. 8 Running time of *DFEKM* and parallel *k*-means in the presence of load imbalance: `kddcup99` dataset

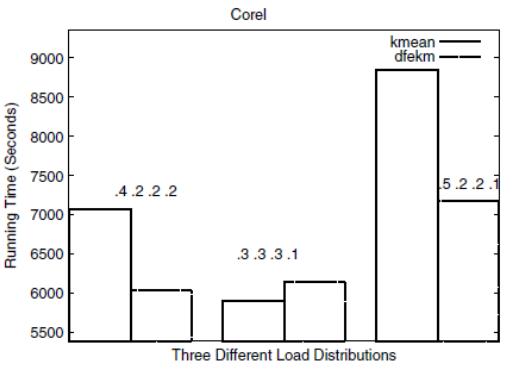
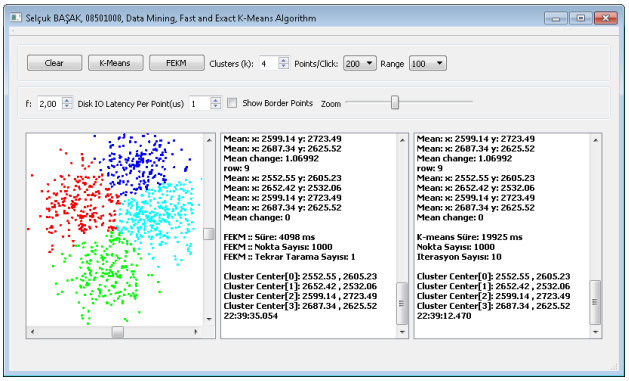


Fig. 9 Running time of *DFEKM* and parallel *k*-means in the presence of load imbalance: `Corel` dataset

7- Uygulama

FEKM'nin performansını değerlendirmek için bir program geliştirdim. Bu program ile iki özellilkli veri seti üzerinde K-Means ve FEKM algoritmalarını parametrik değerlendirmesi yapılabilmektir. Ayrıca, disk erişim gecikmesi ile algoritmaların disk üzerindeki veri setleri üzerinde çalışmasının simulasyonu yapılabilmektedir.



Bu uygulama ile hafızada tutulan veri setleri ile K-Means daha hızlı sonuçlar elde edilmektedir. Ancak nokta başına 1 ms gecikmeler ile bile FEKM, K-Means'e göre 4 kattan daha hızlı sonuçlar vermiştir. Örneğin, nokta başına 1 ms gecikme ile 1000 nokta üzerinde 4 küme ile uygulanan FEKM, tek tarama ile 4098 ms'de tamamlanırken, K-Means ile 10 taramada 19925 ms'de tamamlanmıştır.

8 - Sonuç

Bu çalışmada, K-means ile aynı sonuçları veren ancak daha ay sayıda veri seti taraması gerektiren bir algoritmanın analizi ve değerlendirilmesi yapıldı. Bu algoritma ile disk üzerindeki büyük veri setleri için kümeleme işleminin sonucu aynı kalarak performansının oldukça attırabildiği görüldü. Literatürde Yaklaşık sonuçlar üreten bir çok çalışma olmasına rağmen aynı sonucu daha az tarama ile veren başka bir algoritma yoktur. Algoritmanın temel mantığı, örnekleme ile elde edilen merkez noktaları kullanılarak gerçek merkez noktalarına düzeltmeler yaparak erişilmesi üzerinedir. Yapılan deneysel çalışmalarda orijinal k-means'a göre 2 ila 4.5 kat daha iyi performans elde edilmiştir. Bu çalışmada ayrıca dağıtık FEKM de incelenmiştir. Dağıtık olarak çalışan bu algoritma ile k-means ile aynı sonuç elde edilmekte ve dengesiz yük dağılımlarında, tightly coupled sistemlerde bile paralel K-means'ten daha yüksek başarı elde edebilmektedir.

References

1. Ruoming J.,Anjan G., ·Gagan A. (2005) Fast and exact out-of-core and distributed k-means clustering
2. Berkhin P (2002) Survey of clustering data mining techniques. Technical report, Accrue Software
3. Bottou L, Bengio Y (1995) Convergence properties of the *K*-means algorithms. In: Tesauro G, Touretzky D, Leen T (eds) Advances in neural information processing systems, vol 7. The MIT Press, pp 585–592
4. Bradley PS, Fayyad U, Reina C (1998) Scaling clustering algorithms to large databases. In: Proceedings of the 4th international conference on knowledge discovery and data mining
5. Charikar M, O'Callaghan L, Panigrahi R (2003) Better treaming algorithms for clustering problems. In: Proceedings of the 35th annual ACM symposium on theory of computing
6. Cherikar M, Chekuri C, Feder T, Motwani R (1997) Incremental clustering and dynamic information retrieval. In: Proceedings of symposium of theory of computing
7. Chervenak A, Foster I, Kesselman C, Salisbusy C, Tuecke S (2001) The data grid: towards an architecture for the distributed management and analysis of large scientific datasets. J Network Comput Appl
8. Lopez de Teruel PE, Garcia JM, Acacio M (1999) A parallel algorithm and its application to computer vision. In: Proceedings of PDPTA
9. Dhillon IS, Modha DS (1999) A data-clustering algorithm on distributed memory multiprocessors. In: Lecture notes in computer science, revised papers from large-scale parallel data mining, workshop on large-scale parallel KDD systems. SIGKDD, Springer-Verlag, Berlin Heidelberg New York, pp 245–260 Fast and exact out-of-core and distributed *k*-means clustering 39
10. Domingos P, Hulten G (2001) A general method for scaling up machine learning algorithms and its application to clustering. In: Proceedings of the 18th international conference on machine learning
11. Farnstrom F, Lewis J, Elkan C (2000) Scalability for clustering algorithms revisited. SIGKDD Explor 2(1): 51–57
12. Forman G, Zhang B (2000) Distributed data clustering can be efficient and exact. SIGKDD Explor 2
13. Ghosh J (2003) Scalable clustering methods for data mining. In: Ye N (ed) Handbook of data mining. Lawrence Earlbaum Associates, pp 247–277
14. Guha S, Meyerson A, Mishra N, Motwani R, O'Callaghan L (2003) Clustering data streams: theory and practice. IEEE Trans on Knowl Data Eng 15(3): 515–528
15. Han J, Kamber M (2000) Data mining: concepts and techniques. Morgan Kaufmann
16. Hartigan JA, Wong MA (1979) A *k*-means clustering algorithm. Appl Stat 28:100–108
17. Jain AK, Dubes RC (1988) Algorithms for clustering data. Prentice-Hall International
18. Januzaj E, Kriegel H-P, Pfeifle M (2003) Towards effective and efficient distributed clustering.
In: Proceedings of the ICDM 2003 workshop on clustering large datasets
19. Kargupta H, HuangW, Sivakumar K, Johnson E (2001) Distributed clustering using collective principal component analysis. Knowl Inf Syst 3(4): 422–448
20. Kargupta H, Chan P (1999) (eds) Advances in distributed data mining. AAI/MIT Press 21. Kruengkrai C, Jaruskulchai C (2002) A parallel learning algorithm for text classification.
In: Proceedings of ACM SIGKDD 2002, ACM Press, pp 201–206
22. MacQueen J (1967) Some methods for classification and analysis of multivariate observations.
In: Proceedings of the 5th Berkeley symposium on mathematical statistics and probability,vol 1, pp 281–297
23. Nittel S, Leung KT, Braverman A (2003) Scaling clustering algorithms for massive data sets using data stream. In: Dayal U, Ramamritham K, Vijayaraman TM (eds) Proceedings ofthe 19th international conference on data engineering, March 5–8, 2003, Bangalore, India.IEEE Computer Society
24. OCallaghan L, Mishra N, Meyerson A, Guha S, Motwani R (2002) Streaming-data algorithms for high-quality clustering. In: Proceedings of international conference of data engineering
25. Parthasarathy S, Ogihara M (2000) Clustering distributed homogeneous datasets. In: Proceedings of the 4th European conference on principles of data mining and knowledge discovery, Lecture notes in computer science, vol 1910. pp 566–574. Springer-Verlag,Berlin Heidelberg New York
26. Pelleg D, Moore A (1999) Accelerating exact *k*-means algorithms with geometric reasoning.
In: Proceedings of 5th international conference of knowledge discovery and data mining,pp 277–281
27. Samatova NF, Ostrouchov G, Geist A, Melechko A (2002) RACHET: an efficient coverbased
merging of clustering hierarchies from distributed datasets. Distrib Parallel Databases 11(2):157–180
28. Badoiu M, Har-Pelad S, Indyk P (2002) Approximate clustering via core-sets. In: Proceedings
of the annual ACM symposium on theory of computing