Hadoop MapReduce Algoritmasının Analizi ile Performansa Etki Eden Parametrelerin Tespiti ve Optimize Edilmiş Parametreler ile Hadoop Üzerinde Başarım Artımı

[1]Hüseyin Şarkışla ve [1]Hayrettin Evirgen
[1]Mühendislik Fakültesi, Bilgisayar ve Bilişim Mühendisliği Sakarya Üniversitesi, Türkiye

Özet

MapReduce Kütüphanesi Google tarafından bilişim dünyasına kazandırılan dağıtık mimari üzerinde çok büyük verilerin kolay bir şekilde analiz edilebilmesini sağlayan programlama modelidir. Bu döküman Hadoop MapReduce algoritması iş akışını inceler ve MapReduce işlemlerinin ve konfigürasyon parametrelerinin farklı aşamalardaki farklı kullanımını, ve konfigürasyon parametrelerinin varsayılan değerleri, artıları eksileri ve tavsiye edilen "Konfigürasyon Parametre Modeli" 'ni açıklar. Uygulamaya özgü Konfigürasyon Parametre Modelini oluşturmak için uygulama ortamı düğümler arasında koordinasyonu sağlayan 1 AnaDüğüm (NameNode) ve 4 adet İşçiDüğüm (DataNode) olmak üzere 5 düğümden oluşmuş, her biri 1 gb/s ile haberleşen anahtar (switch) ile birbirine bağlanmış ve Hadoop Küme yapısı oluşturulmuştur. Deneyde yapılan testler ile parametreler için optimum değerler tespit edilmiştir. Amacımız az donanım maliyeti ile ölçekleme yaparak Hadoop MapReduce sistemi için optimum konfigürasyon parametrelerini bulup tavsiye edilen "Konfigürasyon Parametre Modeli" ' ni açığa çıkarmaktır.

Anahtar Kelimeler: Hadoop, Map Reduce, Hdfs, Map Reduce Performans Parametreleri,

With Analysis of Hadoop MapReduce Algorithm Finding Parameters Affecting Performance and Using Optimized Parameters Increasing Throughput on Hadoop Cluster

[1]Hüseyin Şarkışla and [1]Hayrettin Evirgen
[1] Faculty of Engineering, Department of Computer Engineering Sakarya University, Turkey

Abstract

MapReduce Framework is a programming model brought to information world by Google that enables very large data analyzed in easy way on distributed architecture. This study analyses Hadoop Map Reduce algorithm in a way that it describes different phases of MapReduce operations, usage of configuration parameters in the MapReduce job. It explains the configuration parameters, their default values, advantages, disadvantages, and creates a "Configuration Parameter Model" with suggested values in different conditions for this cluster. In order to create Configuration Parameter Model, Hadoop MapReduce cluster is created on environment for experiment which has 5 nodes and has got 1 NameNode which enables coordinating with MasterNode and 4 SlaveNodes. The experiments are made on parameters which is trouble for cluster, optimum parameters values detected made by runnig tests. Our goal is to expose suggested "Configuration Paramater Model" by finding optimum configuration parameters using cluster and by decreasing hardware cost minimum.

Keywords: Hadoop, Map Reduce, Hdfs, Map Reduce Performance Parameters

*Mühendislik Fakültesi, Bilgisayar ve Bilişim Mühendisliği Sakarya Üniversitesi, Türkiye

1. Giriş

2004 yılında Google tarafından geliştirilen MapReduce paralel hesaplama kütüphanesi büyük veri işleme sorunları için etkili bir çözüm haline gelmiştir [11]. İki fonksiyonu olan, map ve reduce basit programlama arabirimleri sayesinde, MapReduce bilişim dünyasında önemli ölçüde birçok büyük veri uygulamalarının tasarım ve uygulamasını kolaylaştırmıştır. Ayrıca MapReduce, yük dengeleme, ölçeklenebilirlik ve hata toleransı dahil olmak üzere yaygın olarak kabul edilen diğer avantajlar da sunmaktadır. Hadoop ise endüstride ve akademik araştırılmalarda yaygın olarak kullanılan MapReduce kütüphanesinin Java programlama dili ile yazılmış açık kaynak uygulamasıdır [1].

Birçok çalışma, farklı düzeylerde veya farklı açılardan Hadoop MapReduce kütüphanesi performansını artırmak için yapılmıştır. Bunlar birkaç kategoriye ayrılmaktadır. Bunlardan ilki iş veya görevlerin daha akıllıca yürütülmesi için bunların sırasını optimize etmek amacıyla, zamanlama algoritmaları tasarımına odaklanmış [12][13]. İkinci grupta özel donanım veya yazılım yardımı ile MapReduce verimliliğinin nasıl artırılması gerektiği için yapılan araştırılmalar yer almakta [14], üçüncü grupta ise belirli bir tipe yönelik özel MapReduce uygulamaları için performans iyileştirmeleri yapılmıştır [15]. Bazı araştırmalar ise yürütme performansını artırmak için optimize edilmiş Hadoop MapReduce yapılandırma ayarları veya parametrelerini keşfetmeye odaklanmıştır.

Birçok araştırmacı Hadoop zamanlama algoritmalarını optimize etmek için çalışmalar yapmışlardır. 2009 yılında, Zaharia heterojen Hadoop kümeleri üzerinde performansını artırmak için LATE (Longest Approximate Time to End) adında özel bir görev zamanlama algoritması önermiştir [12]. Tüm Hadoop kümelerin genel performansını artırmaya yönelik, Hsin-Han dinamik yüklemeden kaynaklanan sorunu çözmek için Load-Aware zamanlayıcı adlı yeni bir algoritma önermiştir [13]. Aynı amaç için Radheshyam tarafından Hadoop kümesi üzerinde çalışan farklı iş tiplerini yürütebilen bir zamanlayıcı tasarlanmıştır. Başka bir çalışma da ise MapReduce performansını artırmak için Localitiy-Aware Görev Zamanlayıcısı (Larts) adında başka bir yaklaşım önerilmiştir [16]. Bu çalışmaların ortak yönleri farklı çalışma durumu için akıllı ve verimli iş ve görev planlaması yaparak Hadoop MapReduce performansını artırmaya yöneliktir. Bu çalışmalara benzer Hadoop MapReduce Algoritması performans iyileştirmesi için birçok çalışma mevcuttur. Ancak çalışmaların birçoğu mevcut kaynakları kullanarak yapılandırılan Hadoop kümesinden ve Hadoop MapReduce kütüphanesinin yapılandırma ayarlarının sisteme özgü hangi değerlerde en uygun değerde olacağı üzerine konularını içermemektedir. Bizim çalışmamamızda ise maliyetleri çok yüksek olan donanımlar almak yerine mevcut olan artık kullanılmayan sıradan bilgisayarlar ile Hadoop kümesi oluşturmaktır. Oluşturulan bu kümenin MapReduce Algoritma analizini, iş akışını yapıp sistemi en uygun hale getirmek için performansa etki eden Hadoop yapılandırma ayar parametrelerini tespit ettikten sonra bu yapıyı genelleştirerek "Optimum Konfigürasyon Parametre Modeli" oluşturmaktır. Uygulama ortamı için sisteme güvenlik duvarı logları yüklenip önce varsayılan parametre ayarları ile test betikleri çalıştırılıp problemler ile karşılaşılmış ve bu yapıdaki Hadoop kümesi

için probleme neden olan parametreler ve değerleri üzerinde deneyler yapılmıştır. Bu yapıyı oluştururken her bir düğümde yapılandırılmış olan Ganglia [17] aracı ile karşılaşılan problemler tespit edilip sistemin kurulmasında probleme neden olabilecek kaynaklar açıklanmıştır. Problemler aşıldıktan sonra logları Linux ortamında istediğimiz şekilde dönüşüm işlerine tabi tutulmuş ve Apache Pig sorguları ile test edilmiştir. Burada dosyalar içerisinde yer alan "IP", "TARİH", "URL1", "URL2", "BROWSER" alanlarını anlamlandırabilmek ve Apache Pig tarafında sütunların ve sütün bilgilerinin doğru bir şekilde oluşması için Sed Unix Komut Düzenleyicisi ile metin dönüşüm işlemleri yapılmıştır.

2. Materyal ve Methodlar

2.1. Hadoop MapReduce Genel Yapısı

MapReduce, İşDenetleyici(JobTracker) ve GörevDenetleyici(TaskTracker) süreçlerinden oluşur. **İşDenetleyici** yazılan MapReduce programının küme üzerinde dağıtılarak çalıştırılmasından sorumludur. Ayrıca dağıtılan iş parçacıklarının çalışması sırasında oluşabilecek herhangi bir problemde o iş parçacığının sonlandırılması ya da yeniden başlatılması da İşDenetleyici'nin sorumluluğundadır.

GörevDenetleyici, İşçiDüğüm'lerin bulunduğu bilgisayarlarda çalışır ve İşDenetleyici' den tamamlanmak üzere iş parçacığı talep eder. İşDenetleyici, AnaDüğüm'ün yardımıyla İşçiDüğüm'ün yerel diskindeki veriye göre en uygun Map işini GörevDenetleyici 'ye verir. Bu şekilde verilen iş parçacıkları tamamlanır ve sonuç çıktısı yine HDFS üzerinde bir dosya olarak yazılarak program sonlanır.

2.1.1 Hadoop MapReduce İş Akışı

2.1.2 Map İşlemleri

Map Süreçleri(Map Processing): HDFS büyük girdi verisini dfs.block.size tarafından kontrol edilen parametre değerine küçük veri bloklarına böler. Varsayılan olarak bu blok değeri 64 mb'tır. Bu veri blokları map görevlerinin girdi verileridir. Her bir map kısmına giden blok sayısı mapred.max.split.size ve mapred.min.split.size parametrelerinin değerlerine bağlıdır. Eğer minimum değer blok boyutundan küçükse ve maksimum değerde blok değerinden büyükse her map kısmına 1 blok gönderilir. Blok verisi girdi formatındaki fonksiyonundaki belirtilen anahtar,değer (key, value) çiftine ayrılır. Map fonksiyonu girdideki her (anahtar,değer) çifti için çalışır ve map fonksiyonundan oluşan çıktı döngüsel olarak bellek tampon kısmına yazılır. Buradaki tampon 100mb varsayılan değerdir ve io.sort.mb özelliği tarafından kontrol edilir.

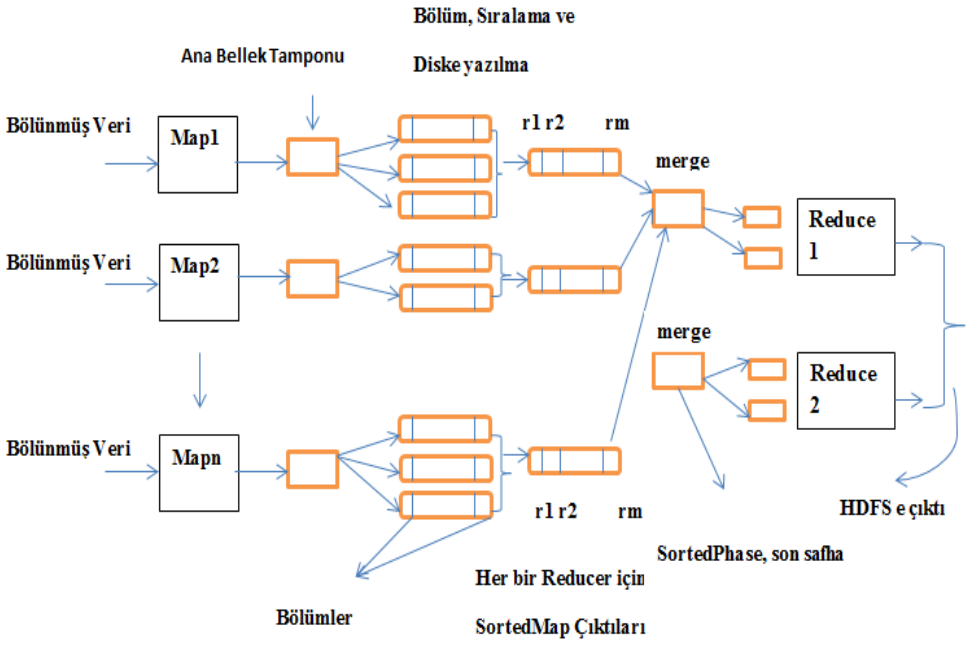
Diske Dökme(Spill): Tampon boyutu io.sort.spill.percent özelliği ile kontrol edilen eşik değerine ulaştığında(varsayılan 0.8) , arka planda izlekler(threads) içeriği tampon bellekten temizleyip diske yazmaya başlar. Diske dökme işlemi başladıktan sonra map kısmında tampon belleğe yazma işlemi devam eder. Bu dökmeler Round-Robin Metodu ile mapred.local.dir özelliği ile

işte belirtilen yere yazarlar ve her tampon bellek eşik değerine ulaştığında yeni bir dökme dosyası oluşturulur.

Bölümleme(Partitioning): Diske yazmadan önce gönderilecekleri Reducer kısmına göre arka planda veriyi böümlere(partition) ayırır.

Sorting: Bellekteki sıralama anahtar değerine göre işleme alınır. Sıralanmış çıktı verisi eğer birleştirme (combine) fonksiyonu tanımlanmış ise ona gönderilir.

Birleştirme(Merging): Map görevi bitmeden önce dökme dosyaları tek bir bölümlenmiş ve sıralanmış çıktı dosyası şeklini alır. Dökme dosya sayısı 3 ten fazla ise Birleştirici (Combiner) devreye girer ve birleştirip tek bir dosyaya dönüştürür. İo.sort.factor özelliği bir seferde maksimum hangi sayıda birleştirme yapılacağını belirtir ve varsayılan birleşme (merge) değeri 10 dur.
Sıkıştırma(Compression): Map çıktısını daha hızlı diske yazma, daha az yer kaplama ve Reducer kısmına giden veri miktarının daha az olmasını sağlamak için diske yazmadan önce sıkıştırma yapabilir. Varsayılan olarak bu değer sıkıştırılmadan yazmaktır. Sıkıştırmayı aktif hale getirebilmek için mapred. compress.map.output değerini "true" yapmak gerekmektedir.
Aşağıdaki diyagram MapReduce işini ve iş içindeki veri akışını farklı safhalardan açıklar.



Şekil 1. MapReduce Algoritmasının küme yapısı üzerinde çalışma prensibi.

2.1.3. Reduce İşlemleri

Kopyalama(Copy): Map görev işlemini tamamladıktan hemen sonra Reducer kısmında karşılık gelen her map görevi çıktı verisini kopyalanmaya başlar. Reduce görevi map çıktı verilerini paralel olarak işleyen 5 izleğe sahiptir ve bu parametre değeri mapred.reduce.parallel.copies

özelliği ile değiştirilebilir. Map çıktı verisi reduce GörevDenetleyici tampon bellek kısmına kopyalanır. Tampon bellek eşik değeri mapred.job.shuffle.merge.percent ve mapred.inmem.merge.threshold özellikleri ile belirlenir. Tampon bellek bu eşik değerine ulaştığında, veri birleştirilir ve oluşturulan dökmeler diske yazılır. Kopyalar diskte toplanırken arka plandaki izlek sıralanmış ve birleştirilmiş dosyaları tekrar birleştirme yapar.

Sıralama(Sort): Aslında bu safha birleşme safhasıdır, çünkü sıralama işlemi map kısmında yapılır. Bu aşama tüm map işlemleri yapıldıktan ve çıktıları kopyalandıktan sonra başlar. Map çıktıları sıralanmış şekli ile birleştirme yapılır. İo.sort.factor varsayılan olarak 10 olduğu için bu sayıdan büyük map sayısı için tur şeklinde çalışır. 40 map çıktımız varsa 4 tur sonunda bu işlemi yapacaktır.

Reduce: Reduce safhasında sıralanmış çıktıdaki her anahtar için reduce fonksiyonu çalışır. Bu aşamanın çıktısı dosya sistemine yani HDFS e direkt olarak yazılır.

2.1.4.Tespit Edilen Parametreler ve İşlev Analizleri

Dfs.block.size : Girdi verinin bölüneceği veri blokları boyutudur.
Mapred.compress.map.output: Çıktı map bölümlerinin sıkıştırılıp sıkıştırılmayacağı parametre değeridir.
Mapred.map/reduce.tasks.speculative.execution: Bir görev, yazılım konfigürasyonundan veya donanımdan kaynaklı nedenlerden dolayı yavaş çalıştığında İşDenetleyici yedek olarak diğer eş değer görevi çalıştırır. Bu "speculative execution" olarak bilinir. Hangisi önce bitirirse diğeri öldürülür.
Mapred.tasktracker.map/reduce.tasks.maximum : Bir GörevDenetleyici için paralel çalışacak maksimum Map/reduce sayısıdır.
io.sort.mb: Map görevi tarafından çıktının sıralama işlemini yaparken kullandığı tampon bellek boyutudur.
io.sort.factor: Map ve Reduce safhasında sıralama işleminde bir kere de birleştirilecek(merge) veri bloğu(stream) sayısıdır.
Mapred.job.reuse.jvm.num.tasks: Bir GörevDenetleyici üzerindeki her JVM için çalışacak maksimum görev sayısıdır.-1 değeri limit olmadığını ve aynı JVM 'nin bir iş için tüm görevler tarafından kullanabileceğini gösterir.
Mapred.reduce.parallel.copies: Map çıktılarını Reducer kısmına kopyalamak için kullanılan izlek(thread) sayısıdır.

3. Deney Ortamı, Deney Sorguları ve Hesaplamalar

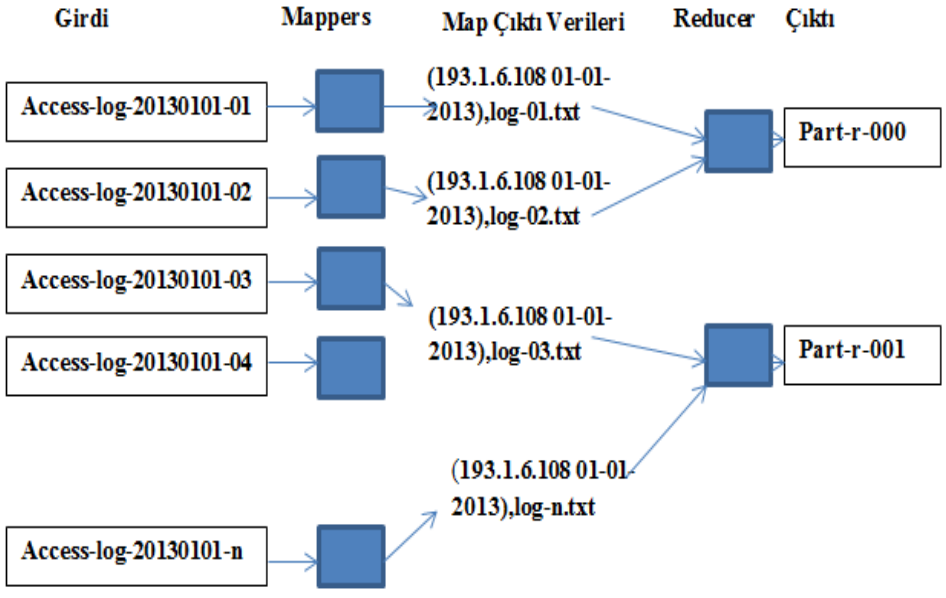
Şekil 2 de optimize edilmiş Hadoop sistemimiz 5 düğümden oluşmuş ve her biri 1 gb/s ile haberleşen switch ile birbirine bağlanmıştır. Hadoop versiyonu 1.2.1 ve sistem Java 1.6.33 üzerinde çalışmakta ve tüm testler için bütün girdi ve çıktılar HDFS' te saklanmıştır. Düğümlerin hepsi 32-bit Ubuntu Linux 12.04 versiyona sahip işletim sistemi üzerinde çalıştırılmış ve her biri Intel Pentium 2.8 Ghz işlemciye 2GB RAM ve 150 GB 7200 RPM disk

özelliklerine sahiptir. Test ortamımızda her biri yaklaşık 102 mb boyutunda Ocak 2013 ayına ait güvenlik duvarı logları oluşan 96 adet 10gb boyutunda bir test verimiz bulunmaktadır. Bu veri içerisinde alanlar '$' işareti ile ayrılmış sırasıyla "IP","TARİH","URL1","URL2","BROWSER" alanları yer almaktadır. Verinin işlenebilmesi için önce HDFS dosya sistemine kopyalanmıştır. Veriyi işleyecek olan test fonksiyonumuz ise Apache Pig Latin dilini kullanarak komut ekranında parametre olarak girilen "IP" ve "TARİH" değerlerine göre; belirtilen IP adresine sahip bilgisayarın belirtilen tarihlerde hangi web sitelerine girmiş olduğunu tespit etmektir.

İstediğimiz IP = "193.1.6.180" ve TARİH= "01-01-2013"değerleri bu şekilde olduğunda betiğimiz;

```
A = LOAD '/input/3ok/' using PigStorage('$') AS
(IP:chararray, Date:chararray,url1:chararray,url2:chararray,browser:chararray);
data = FILTER A BY IP == '193.1.6.108' AND (Date matches '.*31/Dec/2012.*');
STORE data INTO "user/hduser/output"
```

Girdi verilerimiz 96 adet metin dosyasından oluştuğundan dolayı Hadoop MapReduce sistemi AnaDüğüm' ü 96 adet Map ve her biri için GörevDenetleyici oluşturmaktadır. Her bir GörevDenetleyicisinin amacı bulunduğu map bölümünde ilgili filtreleme işlemini gerçekleştirmektir. Her GörevDenetleyici'nin yaptığı filtreleme işleminin sonucu tampon belleklere yazılır. Reducer toplama işlemini yaparken okuma işlemini buradan yapar. Sonuçlar metin dosyası şeklinde tekrar yerel diske yazılır. Şekil 2 bu filtreleme işlemini MapReduce iş akışında daha iyi açıklanmaktadır.



Şekil 2. Logların filtreme işleminin map ve reduce fonksiyonları üzerindeki gösterimi

3.1. Performansa Etki Eden Parametreler ve Analizi

Dfs.block.size : Dosya sisteminin blok boyutudur. Varsayılan olarak 67108864 bayt değerindedir. Deney ortamındaki "Küçük Küme Yapısı(Small Cluster) " için tavsiye edilen değerler için varsayılan blok boyutu çok sayıda map oluşmasına sebep olmuştur. Örneğin;

Girdi Veri boyutu = 20 gb ve dfs.block.size = 64 mb olduğunda minimum map sayısı 384 olur.
Eğer dfs.block.size = 128 mb olursa minimum map sayısı 192, dfs.block.size =256 olduğunda ise minimum map sayısı 96 olur. Küçük küme yapısında(5 node) dfs.block.size yeterince büyük ama tüm küme kaynaklarını kullanabilmek için de mantıklı bir şekilde küçük olmalıdır.

Mapred.compress.map.ouput: varsayılan olarak "false" değerinde yani sıkıştırma kapalı durumundadır. Artıları; daha hızlı disk yazma, disk alanı tasarrufu, mapper tarafından reduce tarafına kısa zamanda veri transferinin gerçekleşmesini sağlar. Eksileri; mapper tarafında sıkıştırma maliyeti ve reducer tarafında da sıkıştırılmış dosyalarının açma maliyetlerinin olmasıdır.
Tavsiye edilen "Konfigürasyon Parametre Modeli" ne göre büyük işler ve büyük küme yapısı için bu değer "true" olmalıdır.

Mapred.map/reduce.tasks.speculative.execution: Belirli görevleri(map/reduce) aktif ve pasif yapar. Artıları; eğer görev işlemi ana bellek veya donanım durumuna göre yavaş ise iş zamanını azaltır. Eksileri; eğer görev işlemi büyük ve karmaşık hesaplama işlemlerinde dolayı yavaş ise iş zamanını artırır. Meşgul map küme yapısında işi biten bir görevin bu meşgul yapıda diğer görevlere katılarak çalışma zamanın azalttığından bu değerin açık olması toplam başarımı artırır.

Tavsiye edilen "Konfigürasyon Parametre Modeli" ne göre küçük yapılarda ve karmaşık olmayan, hesaplama gerektirmeyen işlemlerde değer "Enable", büyük küme yapılarında tüm performansı ve başarımı etkilediğinden bu değer "Disable" olmalıdır.

Mapred.tasktracker.map/reduce.tasks.maximum: Bir GörevDenetleyici için maksimum map/reduce sayısı ve varsayılan olarak 2 dir.
Her bir göreve verilen bellek mapred.child.java.opts özelliği ile belirlenir. Varsayılan olarak Xmx200m(200 mb)

io.sort.mb: Sıralama işlemi için tampon bellek boyutu ve varsayılan olarak 100 mb dır.
Tavsiye edilen "Konfigürasyon Parametre Modeli" ne göre; büyük işler yani map çıktısı verisi çok büyük olan işler için, bu değer artırılmalı ve bu da her bir map görevi için gerekli olan bellek değeri artırılacağı anlamına gelmektedir. Düğümdeki mevcut olan ana bellek miktarına göre ayarlanmalıdır. Ayrıca ne kadar bu değer büyükse o kadar az diske dökme işlemi yani yazma işlemi yapılır.

İo.sort.factor: Veri okuma birleşme (stream merge) faktör değeri ve varsayılan olarak 10 dur.

Tavsiye edilen "Konfigürasyon Parametre Modeli" ne göre; eğer diske yazılan dökme sayısı büyükse yani map sayısı fazla ise bellek miktarı göz önünde bulundurularak bu değerin artırılması gerekmektedir.

Mapred.job.reuse.jvm.num.tasks: tek bir JVM 'yi tekrar kullanma, varsayılan olarak 1 'dir.
Test ortamımızdaki gibi (5 düğüm için 96 map) eğer her bir düğümde çalışacak görev sayısı fazla ise bu değer -1 olmalıdır. Bir görev için yaratılmış olan bir JVM 'yi görevi bittikten sonra diğer bir görevde kullanmak performansta iyileştirmeyi sağlayacaktır.

Mapred.reduce.parallel.copies: Reducer tarafında paralel kopyalama sayısı ve varsayılan olarak 5 'tir.
Tavsiye edilen "Konfigürasyon Parametre Modeli" ne göre; map çıktısı büyük olan büyük işlerde bu değer büyütülebilir ama CPU kullanımı da göz önüne alınmalıdır.

Geçici Alan (Temporary Space): Büyük, orta katman verisi oluşturan işlerde Map çıktısı mapred.local. dir özelliği ile kontrol edilen veri yeterince büyük bir geçici alanda saklanmalıdır. Mapred.local.dir özelliği HDFS sistemindeki disk yolunu belirtir.

3.2. Deney ve Konfigürasyon Modeli

Dfs.block.size değeri varsayılan olarak 64mb'dir.

Model-1.
(Girdi Veri Boyutu *Tampon Bellek Boyutu) / Düğüm Ana Belek Boyutu < Dfs.block.size

Durum-1.
(10000 mb * 100 mb) / 2000 mb < dfs.size.block
500mb < dfs.block.size

Mapred.compress.map.ouput değeri varsayılan olarak "false" durumundadır. Eğer sistem kısıtlı bir disk alanına sahipse disk alanını tasarruflu kullanıp hızlı yazma ve reducer tarafından bilgiye hızlı ulaşması sağlanır. Yalnız CPU özellikleri burada önem taşır. Tek çekirdeğe sahip düğümlerde bu özelliğin açılması önerilmez.

Mapred.map/reduce.tasks.speculative.execution: Meşgul map küme yapısında işi biten bir görevin bu meşgul yapıda diğer görevlere katılarak çalışma zamanın azalttığından bu değerin açık olması toplam başarımı artırır.

Mapred.tasktracker.map/reduce.tasks.maximum bir GörevDenetleyici için maksimum map/reduce sayısı ve varsayılan olarak 2 dir.
Model-2.
Maksimum Görev Sayısı = (Toplam Bellek Boyutu – (GörevDenetleyici + İşçiDüğüm+İşDenetleyici Bellek İhtiyaçları)) / Bir Görev İçin Gerekli Bellek Miktarı

Durum-2.
Eğer düğüm RAM=2 gb ve 1 core CPU sahipse;
Bir görev için maksimum gerekli bellek 500mb ve GörevDenetleyici, İşçiDüğüm ve diğer işlemler için 500+ 500+ 500= 1,5 gb
Maksimum çalışabilecek görev sayısı = (2-1,5)/500mb = 1

Durum-3.
Bir önceki durumda test edilen değerlere oranla;
Eğer düğüm RAM=8 gb ve 8 core CPU sahipse;
Bir görev için maksimum gerekli ana bellek 500mb ve GörevDenetleyici, İşçiDüğüm ve diğer işlemler için 1+1+1= 3 gb
Maksimum çalışabilecek task sayısı = (8-3)/500mb = 10
Sonuç olarak çalışacak map/reduce sayısı bellek kullanımına göre ve görevin hesaplama karmaşıklığına bağlıdır. io.sort.mb değişkeni sıralama işlemi için tampon bellek boyutu ve varsayılan olarak 100 mb 'tır.

Model-3.
(Düğüm Sayısı *Ana Bellek Boyutu)/(Girdi Verisi Boyutu / dfs.block.size) > io.sort.mb

Durum-4.
5*2gb > 10gb/ 64 mb * io.sort.mb
64mb > io.sort.mb

Buradaki durumda bu değer en fazla 64 mb olması önerilir.İo.sort.factor varsayılan olarak bu değer 10 dur.Mapred.job.reuse.jvm.num.tasks varsayılan olarak bu değer 1 dir.

Model-4.
(Girdi Veri Boyutu / dfs.block.size) > Toplam Düğüm Sayısı ise bu değerin -1 alınması önerilir.

Model-5.
10gb/ 64 mb > 5 tir ve mapred.job.reuse.jvm.num.tasks değeri -1 alınması önerilir.

Durum-6.
Mapred.reduce.parallel.copies varsayılan olarak bu değer 5 'tir.
Eğer CPU özellikleri yeterli ise bu değer modeli:

Model-6.
5 < mapred.reduce.parallel.copies < (Girdi Veri Boyutu)/dfs.block.size

Model-7.
Minimum Toplam Disk Boyutu > 3* Analizi Yapılacak Toplam Veri Girdisi + Görev Sayısı * Tampon Bellek Miktarı olmalıdır.

Geçici Alan Miktarı mapred.local.dir değeri en az toplam diskin ¼ katı olmalıdır.

Sonuçlar

Hadoop MapReduce algoritmasını kullanan bir dağıtık sistemin performansı; donanım ekleyip maliyeti artırmadan, girdi verilerinize, yapacağınızın analizin karmaşıklığına ve Hadoop küme yapınızın özelliklerine göre bazı yapılandırma parametrelerini en uygun değerlerine getirerek başarımı gerçekleştirilebildiği tespit edilmiştir.

Bu çalışmada en uygun değerleri yaptığımız testler sonucunda elde edip parametre değerlerini dağıtık sistemimize manuel olarak işledik. Gelecekteki çalışma dağıtık sistem katmanı üzerine bir uygulama ile en uygun değerlerin sistem özelliklerine -yapılandırılan dağıtık disk kapasitesi, girdi verisi, yapılacak analiz türü, her bir düğümdeki bellek miktarları, CPU özellikleri, vb.- göre tespit edilip otomatik olarak parametre değerlerinin değiştirilmesi olacaktır.

Ayrıca gelecekte yapacağımız çalışmalardan bir diğeri de sadece bir sistemden gelen logları yerine yerel veya internet ağında bulunan istenen bir veya birden fazla makinenin(bilgisayar, sunucu, güç kaynağı, güvenlik duvarı, vs.) loglarına otomatik ulaşarak tüm sistemdeki cihazların analizini yapmak olacaktır.

Kaynaklar

[1] Apache hadoop. Website. http://hadoop.apache.org.
[2] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In OSDI, 2004.
[3] I. Elghandour, A. Aboulnaga. Restore: Reusing results of mapreduce jobs. In VLDB, 2012.
[4] Impetus Hadoop Performance Tuning http://www.impetus.com
[5] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava. Building a high-level dataflow system on top of MapReduce: the pig experience. In VLDB, 2009.
[6] H. Herodotou and S. Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. In VLDB,2011.
[7] H. Herodotou, F. Dong, and S. Babu. Mapreduce programming and cost-based optimization? crossing this chasm with starfish. In VLDB, 2011.
[8] F. N. Afrati and J. D. Ullman. Optimizing joins in a mapreduce environment. In EDBT, 2010.
[9] Dev Veri Website. http://devveri.com/hadoop-nedir
[10] C. Lam Hadoop In Action. Apress, 1 edition, June 2011.
[11] J. Dean S. Ghemawat MapReduce:simplified data processing on large clusters Commun. ACM, 51 (1) (2008), pp. 107–113
[12] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, I. Stoica, Improving mapreduce performance in heterogeneous environments, in: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI, 2008, pp. 29–42.

[13] H.H. You, C.C. Yang, J.L Huang, A load-aware scheduler for MapReduce framework in heterogeneous cloud environments, in: Proceedings of the 2011 ACM Symposium on Applied Computing, 2011, pp. 127–132.
[14] S. Zhang, J. Han, Z. Liu, K. Wang, S. Feng, Accelerating MapReduce with distributed memory cache, in: 15th International Conference on Parallel and Distributed Systems, ICPADS, 2009, pp. 472–478.
[15] Y. Becerra Fontal, V. Beltran Querol, P, D. Carrera, et al. Speeding up distributed MapReduce applications using hardware accelerators, in: International Conference on Parallel Processing, ICPP, 2009, pp. 42–49.
[16] R. Nanduri, N. Maheshwari, A. Reddyraja, V. Varma, Job aware scheduling algorithm for MapReduce framework, in: 3rd IEEE International Conference on Cloud Computing Technology and Science, CloudCom, 2011, pp. 724–729.
[17] M. L. Massie, B. N. Chun, D. E. Culler, The ganglia distributed monitoring system: Design, implementation, and experience, Parallel
Computing 30 (2004) 817–840.