한국컴퓨터정보학회 하계학술대회 논문집 제22권 제2호 (2014. 7)

하둡 분산 파일 시스템 기반의 AVL트리를 이용한 동적 복제 관리 기법

류연중°, 윤희용*
°*성균관대학교 정보통신대학
e-mail:{yjryu, youn7147}@skku.edu*°

Dynamic Replication Management Scheme based on AVL Tree for Hadoop Distributed File System

Yeon-Joong Ryu°, Hee-Yong Youn*
°*College of Information and Communication, SungKyunKwan University

● 요 약 ●

클라우드 시스템이 큰 이슈로 떠오르면서 그 기반이 되는 분산 파일 시스템에 관한 연구가 계속되고 있다. 최근 제안된 분산 파일 시스템은 대부분 확장 가능하며 신뢰성이 있는 시스템으로 구성되어 있으며 내고장성(Fault tolerance)과 높은 가용성을 위해 데이터 복제 기법을 사용하며 하둡 분산 파일 시스템에서는 블락의 복제수를 기본3개로 지정한다. 그러나 이 정책은 복제수가 많아지면 많아질수록 가용성은 높아지지만 스토리지 또한 증가한다는 단점이 있다. 본 논문에선 이러한 문제점을 해결하기 위해 최소한의 블락 복제수와 복제된 블락을 효율적으로 배치하여 더 좋은 성능과 부하분산(Load Balancing)하기 위한 기법을 제안한다.

키워드: Load Balancing(부하분산), HDFS, AVL Tree, Data Replication(데이터 복제)

I. Introduction

데이터 복제는 파일 시스템의 데이터 내고장성과 높은 가용성을 위해 널리 사용되고 있다. 최근에는Google file system[1], Ceph file system[2], XFS(Xtreem file system), MooseFS[3], AmazoneS3, HDFS(Hadoop Distributed File System)[4] 등 다양한 분산 파일 시스템이 개발되어 사용되고 있다. 복제수가 많아질수록 내고장성과 가용성이 향상되는 것은 당연한 사실이지만 스토리지 또한 증가한다는 단점이 있다. 이러한 문제를 해결하기 위한 연구들은 계속되고 있는데, HDFS에서는 블락의 기본 복제수를 3개로 지정하고 있다. 본 논문에서는 HDFS 배경에서 최소한의 복제본으로 시스템의 요청을 만족시키기 위한 기법과 최대의 성능과 부하분산(Load Balancing)을 위해 어떻게 효율적으로 복사본을 어떻게 위치 시킬것인지 에 대한 기법을 제안한다.

II. The Proposed Scheme

2.1 시스템 모델링

하둡 분산 파일 시스템은 N개의 독립적이고 여러 다른 종류로 이루어진 데이터 노드 들이 M개의 다른 블락$((b_1, b_2, \ldots, b_{im} \gg N)$들을 저장한다. 본 논문에서는 B 를 데이터 노드 클러스터 안에 블락들의 집합이라 한다. 각 데이터 노드 S_i에는 M_i블락들이 저장되어 있고 $B_i = (b_{i1}, b_{i2}, \ldots, b_{im\,i})$은 S_i에 속하는 블락 들의 집합이다. 블락 b_j는 $b_j = (p_j, s_j, r_j, \tau_j)$속성으로 모델링되며, p_j 는 popularity(블락이 p_j 에 의해 요청되는 확률, 인기), s_j 는 size, $r_j$replica number(복제 갯수), 그리고 τ_j 는 각 블락 b_j의 요청에 대한 접근 지연을 나타낸다. 또한 데이터 노드는 S_i로 표기하고 $S_j = (\lambda_i, \tau_i, f_i, bw_i)$속성으로 모델링 되며, λ_i 는 요청 도착률, τ_i는 평균 서비스 시간, f_i는 실패 확률, bw_i는 각 데이터노드 S_i의 네트워크 대역폭이다.

데이터 노드 S_i 로부터의 블락 b_j요청을 검색 할 때(접근 지연은 τ_i 보다 작은), 대역폭은 성능을 보장하기 위해 S_j/τ_i 로 세션에 할당되어야 한다. 물론 S_i의 전체 대역폭은 bw_i보다 크지 않은 요청을 서비스 해야 한다.

$$bw_i \geq \sum_{j=1}^{C_i} \frac{s_j}{\tau_j} \quad (1)$$

2.2 가용성(Availability)

이번 세션에서는 앞으로 사용하게 될 노드, 블락, 파일의 가용

(Available)과 비가용(Unavailable) 대해 정의 한다.

Node Available : 도달 가능한 데이터 노드의 이벤드를 NA 표기하며, $P(NA)$ 는 Node Available 의 확률을 나타낸다.

Node Unavailable : 도달 할 수 없는 데이터 노드의 이벤트를 로 표기하며, 는 Node Unavailable 의 확률을 나타낸다. 그리고 = 1-$P(NA)$이다.

Block Available : 블락 B_i의 가용성을 BA_i로 표기하며, $P(BA_i)$는 블락 B_i의 가용에 대한 확률을 나타낸다.

Block Unavailable : 블락 B_i의 비가용을 로 표기하며, 는 블락 B_i의 비가용에 대한 확률을 나타낸다. 그리고 = 1-$P(BA_i)$이다.

File Available 과 File Unavailable 또한 위의 방식대로 $P(FA)$과 나타낸다. 또한 = 1- $P(FA)$이다.

파일 F가 m개의 블락으로 나누어진다면 $\{b_1, b_2, \ldots, b_m\}$으로 나타낼 것이고, 서로 다른 데이터 노드들 안에 분산저장 될 것이다. 만약 데이터노트들에 저장되어 있는 블락 들이 Unavailable 상태이라면 당연히 블락 b_j의 복제들 r_j 또한 Unavailable 상태일 것 이고, 블락 b_j의 Unavailable 상태 확률은 아래와 같고,

$$P(\overline{BA}_i) = P(\overline{NA}_1 \times \overline{NA}_2 \times \ldots \times \overline{NA}_{r_j})$$

데이터 노드들은 독립적이기 때문에 아래와 같은 공식을 얻을 수 있다.

$$P(\overline{BA}_i) = P(\overline{NA}_1) \times P(\overline{NA}_2) \times \ldots \times P(\overline{NA}_{r_j}) = \prod_{i=1}^{r_j} f_i \quad (2)$$

모든 파일F 을 검색하기 위해선 모든 m블락들을 가져야 한다. 하나의 블락 이라도 Unavailable 상태라면 File Unavailable 상태를 야기할 수 있고 아래와 같은 식으로 표현 한다.

$$P(\overline{FA}) = P(\overline{BA}_1 \cup \overline{BA}_2 \cup \ldots \cup \overline{BA}_m) \quad (3)$$

$$\sum_{j=1}^{m} P(\overline{BA}_j) - \sum_{1 \le j < k \le m} P(\overline{BA}_j \cap \overline{BA}_k) + \ldots + (-1)^{m-1} P(\overline{BA}_1 \cap \overline{BA}_2 \cap \ldots \cap \overline{BA}_m)$$

식(2)을 식(3) 에 안에 대입하면 아래와 같고,

$$(\overline{FA}) = \sum_{j=1}^{m} (-1)^{j+1} C_m^j (\prod_{i=1}^{r_j} f_i)^j$$

결과적으로 파일 F의 가용성은 아래와 같다.

$$P(FA) = 1 - P(\overline{FA}) = 1 - \sum_{j=1}^{m} (-1)^{j+1} C_m^j (\prod_{i=1}^{r_j} f_i)^j$$

파일 F 에 기대된 가용성이 사용자에 의해 A_{expect}라고 정의하고 요청된 가용성을 만족하기 위해선 아래와 같이 표현된다.

$$1 - \sum_{j=1}^{m} (-1)^{j+1} C_m^j (\prod_{i=1}^{r_j} f_i)^j \ge A_{expect} \quad (4)$$

네임 노드는 식(4)로 평균 데이터 노드 실패율과 함께 기대된 가용성을 만족시키기 위해 최소한의 복제 개수$r_{\min}$ 값을 계산한다.

2.3 블락 확률(Blocking Probability)

데이터 복제 수를 결정한 후에 가용 요청을 만족시키기 위해 어떻게 복제 데이터들을 효율적으로 위치시켜 최고의 효율과 부하분산(Load Balancing)할 것 인지를 이번 세션에서 소개한다.

블락 확률(Blocking Probability)은 데이터 노드간에 복제 블락들을 배치하고 비대칭 접근을 줄이기 위한 기준으로 사용된다.

$\frac{P_j}{r_j}$ 는 블락b_j 접근할 확률이고, $\sum_{j=1}^{M_i} \frac{P_j}{r_j}$ 는 데이터 노드 S_i에 접근할 확률이다.

본 논문은 포아송 분포(Poisson distribution)의 전체 도착률 λ에 따라 데이터 노드 클러스터에 도착하는 요청들을 가정해 보았다. 데이터 노드 S_i 에 요청 도착률은 아래와 같은 식으로 표현된다.

$$\lambda_i = \sum_{j=1}^{M_i} \frac{P_j}{r_j} \lambda$$

데이터 노드 S_i 의 평균 서비스 타임은 아래와 같다.

$$\tau_i = \frac{1}{M_i} \sum_{j=1}^{M_i} \tau_j$$

요청을 받은 후에 데이터 노드 S_i평균 τ_i 지속 시간 동안 서비스를 한다. 데이터 노드 S_i이 C_i 세션에 서비스 한다면 추가적인 요청에 대한 서비스를 할 수 없고 요청이 들어와도 거절하게 되거나 블락 된다. 데이터 노드 S_i은 $M/G/C_i$ 도착률 λ_i 와 서비스율 $1/\tau_i$를 통해 모델링 될 수 있다[5]. 이를 통해 데이터 노드 S_i의 블락 확률(Blocking Probability)은 아래와 같이 표현된다.

$$BP_i = \frac{(\lambda_i \tau_i)^{c_i}}{c_i!} \left[\sum_{k=0}^{c_i} \frac{(\lambda_i \tau_i)^k}{k!} \right]^{-1} \quad (5)$$

식(5)로 블락 확률(Blocking Probability) BP_i를 통해 워크로드의 강도와 서로 다른 λ_i 와 τ_i 값에 의한 데이터 노드 S_i의 성능을 알 수 있다.

AVL트리는 균형 잡힌(Balanced) 이진탐색 트리로서 각각의 노드 마다 왼쪽과 오른쪽의 높이 차이에 대한 정보를 가지며 부분 트리의 높이 차이가 1보다 크지 않은 성질을 갖는다. 균형 잡힌 AVL트리는 n개의 원소가 있을 때 평균 O(logn)의 시간복잡도로 검색, 삽입, 삭제를 할 수 있으며 최악의 경우에도 O(logn)의 시간 복잡도를 갖는다. 이에 본 논문에서는 네임 노드가 AVL 트리를 사용하여 데이터 노드 들의 블락 확률(Blocking Probability) BP_i값을 통해 Fig.1 과 같이 정렬한 후, 새로운 후보 데이터 노드가 들어오면 네임 노드는 빠르게 AVL 트리를 검색하고 가장 낮은 블락 확률(Block Probability)의 데이터 노드 ID를 알려준다. 이런 메커니즘으로 검색은 매우 빠르며 컴퓨팅 워크로드는 극도로 낮아지게 된다.

각 데이터 노드의 블락 확률은 데이터 노드 쪽에서 지역적으로 계산되며 네임 노드에 주기적으로 업데이트되는데 이것은 네임노트의 관리 워크로드를 줄여 줄수가 있다. 업데이트된 블락 확률 값을 통해 네임 노드는 빠르게 AVL 트리의 데이터 노트 ID를 재배치 하고 복제 데이터들의 배치를 결정 할 수 있다.

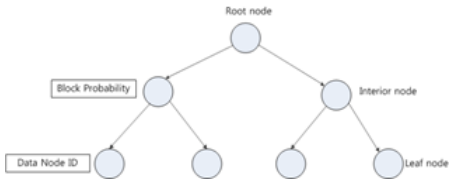

Fig. 1. DataNode AVL Tree

III. Performance Evaluation.

본 논문은 1개의 네임 노드와 20개의 데이터 노드로 구성된 클러스터로 구축되었다. 각 노드는 IntellPentium 4CPU 2.8GHz, 2GB memory그리고 80GB SATA disk 를 사용하였다. 그리고 RedHat AS4.4에 kernel 2.6.20의 운영체제와 하둡 0.16.1버전 그리고 자바 1.6.0 버전에서 실험하였다.

본 논문에서는 한 블록당 A_{expect} 값을 0.8로 유지하면서 실패율을 각각 0.1과0.2 두 경우를 두고 복제본 개수를 측정해 보았다. 아래 Fig.2 에서 처럼 처음 복제 개수가 증가하다가 어느 지점에서부턴 복제 개수가 유지되는 것을 볼 수 있었다. 이것은 만약 복제 개수가 가용성 요청을 만족시키지 못한다면 복제 개수를 늘리게 되고 반대로 최소한의 복제 개수로 만족을 시켜준다면 복제 개수를 유지시켜 비용을 절약하는 것을 볼 수 있다.

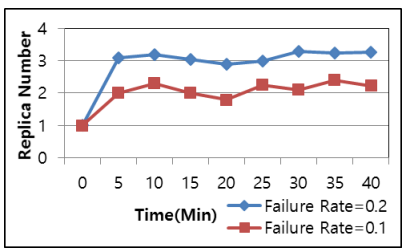

Fig. 2. Compare failure ratio of 0.1 and 0.2

각각 다른 분산 워크로드에서의 성능을 비교하기 위해 본 논문에서는 인기값(Popularity)을 10% 에서 100%로 두고 실험을 하였으며 제안하는 ADRM(AVL Tree Dynamic Replication Management)과 기본 복제 관리로 쓰이는 HDFS의(HDRM)사이에서 비교 하였다. HDRM은 정적으로 시스템간의 복제 수를 유지하는 복제 배치정책을 채택하고 있다. 본 논문에선 Aexpect 값은 0.8 평균 실패율은 0.1, 시스템은 2개의 복제 수를 유지하도록 한다. 실험 결과는 Fig.2 와 같다.

Fig.3 에서 볼 수 있듯이 인기(Popularity)가 증가 함에 따라 ADRM과 HDRM보두 접근 지연이 줄고 있다. 그러나 인기가 낮을 때ADRM의 성능이 HDRM에비해 훨씬 좋은 결과를 볼 수 있다. 그 이유는 ADRM은 동적으로 데이터 노드 간 워크로드를 재분산 처리하고 데이터노드 용량과 워크로드 변경을 통해 복제데이터의 위치를 조정하기 때문이다. 또한 접근 도착률의 0.2에서 0.6으로 증가 했을 때 ADRM의 성능이 HDRM보다 좋은 것을 볼 수 있다.

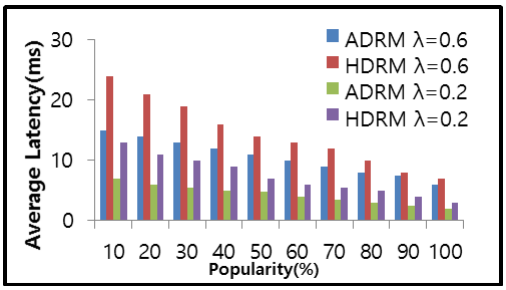

Fig. 3. Effect of popularity and access arrival rate

데이터 노드들 간 복제 배치 정책은 부하분산(Load Balancing)에 큰 영향을 준다. Fig.4 에서의 실험을 보면 ADRM 과 HDRM의 사용률을 비교하고 있다. Fig.4 에서 볼수 있듯이 클러스터의 평균 시스템 이용률과 비교하여 각 데이터 노드의 시스템 이용률의 차이를 보여주고 있으며 위의 실험은 본 논문이 제안하는 ADRM이 더 효율적이고 개선된 성능과 가용성 그리고 부하분산을 해주는 것을 보여준다.

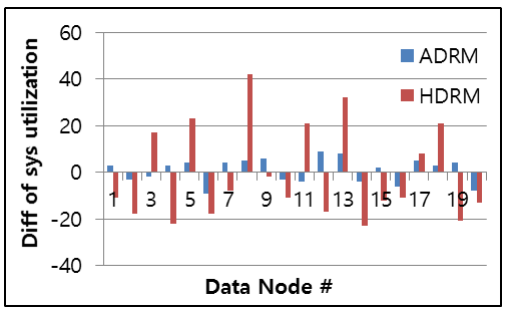

Fig. 4. System utilization among data nodes

IV. Conclusions

본 논문에서는 AVL트리를 이용한 동적 복제 관리 기법을 제안하여 최소한의 복제본으로 시스템의 요청을 만족시키기 위한 기법과 최대의 성능과 부하분산(Load Balancing)을 위해 어떻게 효율적으로 복사본을 위치 시킬것인지 에 대한 기법을 제안하였다. 제안된 기법의 실험결과 기존의 HDFS의 성능보다 인기(Popularity)에 따른 평균 접근 지연 도착률이 낮아졌고, 데이터 노드들 간의 부하분산(Load Balancing)에서도 높은 성능을 보였다.

ACKNOWLEDGMENT

본 연구는 BK21+사업, 한국연구재단 기초 연구사업 (2014R1A1A2040257), (2014R1A1A2060398), 미래부가 지원한 2014 년 정보통신·방송(ICT) 연구개발 사업의 연구 결과로 수행되었음

References

[1] S. Ghemawat, H. Gobioff, and S.T. Leung, “The Google File System,” In Proc. Of the 19thACMSymp.OnOperatingSystemsPrinciples,2003.

[2] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, “Ceph: A Scalable, High-Performance Distributed File System,” In Proc. of the 7thSymposiumonOperatingSystemsDesignandImplementation,2006.

[3] MooseFS, http://www.moosefs.org.

[4] D. Borthakur, “The Hadoop Distributed File System: Architecture and Design,” The Apache Software foundation, 2007.

[5] Dan Feng, Lingjun Qin. Adaptive Object Placement in Object-Based Storage Systems with Minimal Blocking Probability. Proceeding of the 20thinternationalconference(AINA’06),2006