클러스터 파일 시스템의 내용 기반 부하 분산 알고리즘[1)]

장준호, 박성용
서강대학교 컴퓨터학과
jj@dcclab.sogang.ac.kr, parksy@sogang.ac.kr

A Content-based Load Balancing Algorithm for Cluster File System

Jun-Ho Jang, Sung-Yong Park
Dept. of Computer Science, Sogang University

요 약

메타데이타에 대한 접근이 특정 디렉토리에 집중되며 메타데이타 연산마다 다른 계산량을 가지는 클러스터 파일 시스템의 특성상 메타데이타 서버 간 부하의 불균형과 과부하가 발생한다. 따라서 클러스터 파일 시스템의 성능을 결정짓는 중요 요소인 메타데이타 서비스의 성능을 위해서는 메타데이타 서버들의 과부하 상황에 대처할 수 있는 합리적인 부하 분산 기법이 필수적이다. 메타데이타 공간을 분할하여 담당 영역만을 관리하는 비대칭 메타데이타 서버를 위해 본 논문은 클라이언트 요청의 내용을 분석하여 담당 메타데이타 서버를 결정하고 해당 연산의 종류에 따라 단순 검색, 메타데이타 중복 저장(replication), 또는 메타데이타에 대한 로깅(logging)을 수행하는 내용 기반의 부하 분산 알고리즘을 제시하였다.

1. 서 론

고성능 컴퓨터에 대한 벤치마크 기관인 SPEC의 연구에 따르면 클러스터 파일 시스템에 대한 트래픽의 60%이상이 메타데이타에 대한 요청(request)에 집중되고 있다[1]. 최근 클러스터 파일 시스템은 고성능, 가용성, 확장성을 이유로 여러 대의 메타데이타 서버(metadata server; 이하 MDS)를 이용하여 클러스터를 구축하려는 시도를 하고 있다[2][3]. 공유 저장 장치를 이용하는 대칭형(symmetric) 클러스터링 기법에 비해 성능적으로 뛰어난 비대칭형(asymmetric) 기법이 현재 MDS 연구의 주류를 이루고 있다. 비대칭 클러스터링 기법은 메타데이타 공간을 분할하여 각 MDS에게 담당 영역을 할당한다.

클러스터 파일 시스템의 비대칭 MDS를 위한 대표적인 연구로 Vesta[2]와 LH3[3]가 있다. 두 기법은 파일의 전체 경로를 입력값으로 가지는 해쉬 함수를 이용, 메타데이타를 다수의 MDS에 분산 저장한다. 해쉬 함수는 메타데이타의 직접 접근을 보장하지만, 디렉토리 관리의 어려움이 있다. Vesta는 파일들의 묶음 기능만을 제공할 뿐 디렉토리 접근 권한 관리는 제공하지 않는다. LH3는 빠른 하위 디렉토리 검색을 위해 메타데이타간의 디렉토리 계층 구조를 추가적으로 유지하고 있다. LH3는 글로벌 로깅 기법을 이용, 디렉토리 명 변경에 대한 정보를 관리한다. 디렉토리 수정에 대한 요청을 MDS는 로깅하고, 로그 정보는 모든 MDS간에 브로드케스트 통신을 이용하여 공유한다. 특정 메타데이타에 대한 정보를 요청하면 MDS는 로깅 정보를 검색해 해당 메타데이타의 실제적인 위치를 파악한다. 실제적인 메타데이타의 이동은 클라이언트 요청이 도착했을 때나 후위 작업(background job)으로 수행한다.

Vesta, LH3와 같은 기존 연구는 메타데이타 생성 시점에 담당 MDS를 결정지음으로서 부하를 실시간으로 반영하지 못한다. 또한 디렉토리 명 변경으로 인한 하위 파일과 디렉토리의 메타데이타에 대하여 재배치, 그리고 하위 디렉토리 검색시 다수의 MDS 접근을 필요로 하며, 이는 클라이언트 요청의 응답 시간 지연을 발생시킨다. 마지막으로 LH3의 글로벌 로깅 기법은 디렉토리 삭제시 적용할 수 없는 단점이 있다. 또한 메타데이타에 대한 접근이 특정 디렉토리에 집중되며 메타데이타 연산마다 다른 계산량을 가지는 클러스터 파일 시스템의 특성상 MDS간 부하의 불균형과 과부하가 발생한다. 따라서 비대칭 MDS 클러스터를 위한 합리적인 부하 분산 기법이 필수적이다.

레이어3(Layer3)와 레이어4(Layer4) 기반의 부하 분산 기법은 동일한 성격의 요청이 모든 서버들에게 걸쳐 나눠지는 특징이 있다. 메타데이타 수정이 빈번한 클러스터 파일 시스템 특성상 이 기법들은 병행 제어(concurrency control)에 대한 부담을 가중시키며 비대칭 클러스터에게 적용하기에는 무리가 있다. 이와 같은 이유로 특정 MDS의 과부하 상황 제거, 응답 시간의 최소화, 그리고 병행 제어에 대한 부담을 줄여 수 있는 내용 기반(content-based)의 부하 분산 기법이 연구되어야 한다. 본 논문은 구조가 간단하고 성능이 안정적이라고 평가받는 부하 분배기(dispatcher)[4]를 이용, 클러스터 파일 시스템 연산(operation)의 특성을 반영한 기법을 연구하였다.

본 논문의 나머지 구성은 다음과 같다. 2절에서는 본 논문의 부하 분산 알고리즘을 구체적으로 제시한다. 3절에서는 성능을 평가하고 마지막으로 4절에서 결론을 내린다.

2. 내용 기반의 부하 분산을 지원하는 메타데이타 관리 기법

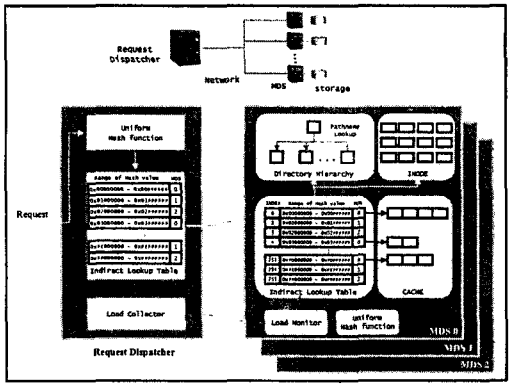
2.1 비대칭 메타데이타 서버 구조

본 논문이 제시하는 비대칭 MDS 클러스터를 위한 동적인 부하 분산 알고리즘은 요청 분배기, 다수의 메타데이타 서버, 해쉬 함수, 간접 검색 테이블(Indirect Lookup Table), 디렉토리 계층 구조, 로컬 디스크, 그리고 캐쉬를 이용하여 동작하며 그 구조는 <그림 1>과 같다.

클라이언트의 모든 요청은 부하 분배기로 전달되며 부하 분배기는 요청을 해쉬 함수와 간접 검색 테이블을 이용하여 담당 MDS를 결정한다. 부하 분배기는 메타 데이터 연산의 종류에 따라 요청을 전체 MDS에게 브로드캐스팅 또는 담당 MDS에게만 전달한다. 비대칭 MDS는 부하 분배기와 동일한 해쉬 함수와 간접 검색 테이블 정보를 가지며 연산의 종류에 따라 담

1) 본 연구는 한국과학재단 목적기초연구(R01-2003-000-10627-0) 지원으로 수행되었음.

당 MDS와 비담당 MDS들의 수행 동작이 결정된다. MDS는 성능 향상을 위하여 inode에 대하여 캐쉬를 사용한다. 또한 연산 시간이 긴 디렉토리 경로 검색 또는 권한 검사의 속도 향상을 위하여 디렉토리 계층 구조를 따로 관리하고 있다.



<그림 1> 비대칭 메타데이타 서버 구조

2.2 파일 시스템 연산에 따른 MDS의 동작 과정

본 논문은 리눅스 VFS의 inode 연산[5]을 읽기 연산, 디렉토리 관련 쓰기 연산, 그리고 파일 쓰기 연산으로 분류하였다.

읽기 연산은 간접 검색 테이블을 통하여 담당 MDS를 선택한 후 메타데이타 검색을 수행한다. 그러나 inode의 "last access field" 필드로 인하여 일부 읽기 연산은 필드 수정을 수행하여야 한다. 본 논문은 읽기 연산을 두 종류로 분류하였으며 후자의 경우 비담당 MDS에서 메모리 로깅(logging)을 수행한다.

MDS는 디렉토리 관련 쓰기 연산에 대하여 inode 연산과 더불어 디렉토리 계층 구조의 수정을 수행한다. 비대칭 MDS 경우 디렉토리 계층 구조의 변동은 MDS간 대량의 메타데이타 이동을 필요로 하기 때문에 수행 시간이 길다. 그러나 본 논문은 모든 MDS들이 동시에 수행한다. 이로 인하여 비담당 MDS에 추가적인 부하가 유발되나, MDS간의 메타데이타 이동을 제거하여 메타데이타 연산에 대한 응답 시간을 단축시켰다.

파일 관련 쓰기 연산은 디렉토리 쓰기 연산과 달리 디렉토리 계층 구조의 수정을 요구하지 않음으로 짧은 연산 시간을 갖는다. 그러나 디렉토리 쓰기 연산과 같이 메타데이타 일치성을 위하여 모든 MDS들이 동시에 진행한다.

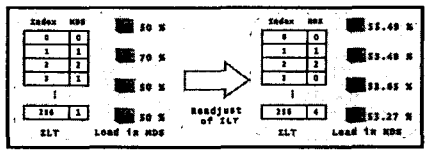
<표 1> 메타데이타 연산의 수행 시간과 부하량

	수행 시간	부하량
디렉토리 정보 검색	O (1)	O (1) or O (N)
파일 정보 검색	O (1)	O (1) or O (N)
디렉토리 경로 검색	O (depth of tree)	O (depth of tree)
디렉토리 생성	O (depth of tree)	O (N*depth of tree)
디렉토리 삭제	O (depth of tree)	O (N*depth of tree)
디렉토리 명 변경	O (depth of tree)	O (N*depth of tree)
디렉토리 정보 수정	O (1)	O (N)
파일 생성	O (1)	O (N)
파일 삭제	O (1)	O (N)
파일 명 변경	O (1)	O (N)
파일 정보 수정	O (1)	O (N)

MDS의 개수를 N이라 할 때 제안 기법의 연산 별 수행 시간과 전체 MDS에서 발생하는 부하량은 <표 1>과 같다.

2.3 내용 기반 부하 분산 알고리즘

본 논문이 제안하는 알고리즘의 핵심은 <그림 2>와 같이 검색 테이블의 재조정을 통하여 모든 MDS가 평균 부하량에 근접한 부하량을 가지도록 하며 검색 테이블 엔트리의 담당 MDS의 변경이 최소화되도록 하는 것이다.



<그림 2> 간접 검색 테이블 조정의 예

모든 MDS의 평균 부하량 계산한 후 mds_i에 대하여 평균 부하량에서 현재 부하량은 뺀 $Extra(mds_i)$를 구한다. 그 값이 음수인 경우는 담당 엔트리의 개수를 줄여 부하를 떨어뜨려야 할 MDS이다. 현재 부하 $Load(mds_i)$를 mds_i가 담당하고 있는 엔트리의 개수 e로 나눈 값이 엔트리당 부하인 $Load_e(mds_i)$가 된다. $Extra(mds_i)$가 음수인 mds_i가 줄여야 할 최대 엔트리의 개수 $MAX_i(n)$은 <식 1>을 만족시키는 정수 n 중에서 최대값이 된다.

$$Extra(mds_i) - Load_e(mds_i) \times MAX_i(n) \le 0 \qquad \text{<식 1>}$$

부하가 평균치 이하인 mds_j는 평균치 이상인 mds_i의 담당 엔트리를 대신 할당받게 된다. 즉 $Extra(mds_j)$가 양수인 mds_j는 $Extra(mds_i)$가 음수인 mds_i의 담당 엔트리로부터 가져오게 되며, 가져올 엔트리의 개수 $MAX_j(s)$는 <식 2>와 같다.

$$Extra(mds_j) + \frac{1}{2} Load_e(mds_i) - Load_e(mds_i) \times MAX_j(s) \ge 0 \qquad \text{<식 2>}$$

3. 성능 평가

본 절에서는 프로세서 기반 이산 사건 시뮬레이터인 Mesquite Software사의 CSIM19[6]를 사용하여 제안 기법의 성능을 측정하였다. 시뮬레이션은 20000 millisecond 동안 클라이언트 요청을 집중적으로 발생시켜 진행하였으며 성능 판단을 위하여 두 종류의 측정을 실시하였다. 첫번째는 각 MDS의 부하량이고 두번째는 각 클라이언트 요청에 대한 응답 시간(response time)이다.

클러스터 파일 시스템의 읽기와 쓰기 연산의 평균적인 비율이 9:1이다[7]. 실험은 평균적인 연산 비율에 대한 성능 측정과 함께 쓰기 연산 비율이 증가할 경우의 성능 변화를 측정하였다. 마지막으로 부하 분배기의 성능에 따른 알고리즘의 성능 변화를 측정하였으며, 그 결과를 기존 연구와 비교하였다.

3.1 시뮬레이션 모델

<표 2> 시뮬레이션 파라미터

the number of MDS	8
metadata size	256 Byte
mean memory cache search time	0.155 msec for 10 Mbyte
memory cache hit ratio	90 %
disk access time	1.561 msec for 1 metadata
network transfer time	0.209 msec for 1 metadata

시뮬레이션을 위한 MDS는 인텔 펜티엄III-800Mhz Dual SMP CPU, PC100 메모리, 5600 RPM(Ultra ATA-66)의 HDD와 리눅스 kernel2.6을 OS로 사용하는 서버를 모델링하였다.

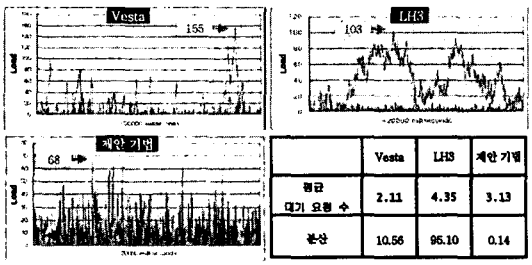
클러스터 파일 시스템의 일반적인 환경과 리눅스 커널의 추가적인 비용을 고려한 시뮬레이션 파라미터는 <표 2>와 같다.

3.2 읽기와 쓰기 연산 비율이 9:1 일때 성능 비교

<그림 3>의 그래프는 시간의 흐름에 따라 각 MDS에서 수

행을 기다리는 대기 요청의 개수 변화를 나타내며, 그래프의 선은 MDS별 대기 요청 수를 의미한다. Vesta와 LH3는 특정 MDS의 대기 요청 수가 다른 MDS에 비하여 높고 그 기간이 지속됨을 보였다. 이것은 특정 MDS에게 부하가 집중되고 있음을 뜻한다. 그러나 제안 기법은 짧은 기간을 보였다. 즉 제안 기법이 다른 기법에 비하여 부하를 효과적으로 분산하였다.

<그림 3>의 표를 참조하면 보다 명확히 파악할 수 있다. 제안 기법의 평균 대기 요청 수가 3.13으로 Vesta(2.11)보다 높지만 MDS 간 분산이 0.14로 Vesta(10.36)보다 분포가 균일함을 알 수 있다. 즉 제안 기법이 평균 요청 개수를 상향 평준화하여 특정 MDS의 최대 요청 개수를 낮췄다.



	Vesta	LH3	제안 기법
평균 대기 요청 수	2.11	4.35	3.13
분산	10.56	96.10	0.14

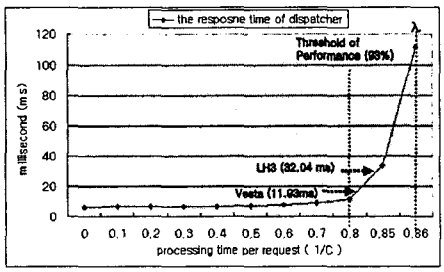
<그림 3> 연산 비율이 9:1일때 성능(대기 요청 수) 측정

3.3 부하 분배기의 성능에 따른 제안 기법의 성능 변화

큐잉 이론 (Queuing Theory)[8]에 따르면 부하 분배기의 응답 시간은 $R=1/(C-\lambda)$이다(C는 부하 분배기의 처리 용량, λ는 클라이언트 요청의 발생 비율). 즉 λ가 고정되었을 때 응답 시간은 부하 분배기의 용량 C에 의하여 결정된다.

본 절에서는 3.2 절의 실험 환경(연산 비율이 9:1)을 이용하여 부하 분배기의 용량이 제안 기법의 성능에 끼치는 영향을 실험하였다. 클라이언트의 요청이 λ(0.86)의 비율로 부하 분배기에 도착할 때, 클라이언트의 요청 당 처리 시간(1/C)에 따른 제안 기법의 응답 시간은 <그림 4>와 같이 변화한다. 1/C가 0.8 이하일 때는 점진적으로 응답 시간이 증가하고 있으며, 0.8을 넘어서는 시점에서는 기하 급수적으로 증가하고 있다. 또한 1/C 이 0.8 이하일때 제안 기법의 성능은 기존 연구 (Vesta, LH3)보다 우수한 성능을 유지하고 있음을 알 수 있다.

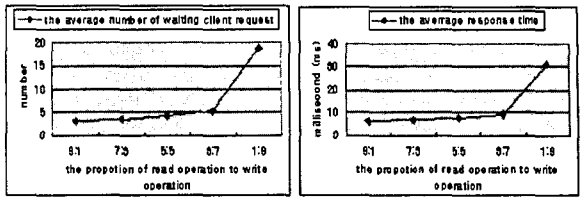
정리하면, 본 실험 환경하에서 λ에 대한 1/C가 93% 수준일 때까지 제안 기법의 응답 시간은 점진적으로 증가하였으며 기존 연구와 비교하여 우수한 응답 시간을 가짐을 보였다.



<그림 4> 부하 분배기의 성능에 따른 제안 기법의 성능(응답시간) 변화

3.4 연산 비율에 따른 제안 기법의 성능 변화

<그림 5>는 읽기와 쓰기 연산의 비율이 9:1, 7:3, 5:5, 3:7, 1:9일때 성능을 측정한 것이다. 부하 분배기의 1/C는 0으로 가정하였다. 왼쪽은 MDS에서 수행을 기다리는 클라이언트 요청의 평균 개수에 대한 그래프이고, 오른쪽은 요청에 대한 평균 응답 시간을 측정한 것이다. 쓰기 비율이 증가함에 따라서 디렉토리 계층 구조 수정에 대한 부담으로 제안 기법의 성능은 저하되었으며, 쓰기 비율이 9인 경우의 성능은 비율이 1인 경우와 비교하여 80% 이상 저하됨을 보였다.



<그림 5> 연산 비율에 따른 제안 기법의 성능(대기 요청 수, 응답 시간) 변화

4. 결론

본 논문에서는 메타데이타 서비스의 성능에 영향을 미치는 파일 시스템 연산의 특징을 분석하여 클러스터 파일 시스템의 비대칭 MDS에 적합한 내용 기반의 부하 분산 알고리즘을 제안하였다.

성능 측정 결과, 제안 알고리즘이 부하의 효율적인 배분을 통하여 MDS간 부하량을 균일화 시킬 수 있음을 보였다. 그러나 쓰기 연산의 비율을 증가시킬 경우 제안 기법의 성능은 저하되었다. 제안 기법의 성능을 결정하는 중요 요소는 부하 분배기이다. 본 논문의 실험 환경에서 λ에 대하여 1/C가 93% 수준일 때까지는 제안 기법의 응답 시간이 점진적으로 증가하였으며 기존 연구와 비교하여도 우수한 응답 시간을 가졌다.

참고문헌

[1] SPEC, *"SFS 3.0 Documentation Version 1.0"*, Standard Performance Evaluation Corporation, 2001.

[2] Peter F. Corbett et al., *"The Vesta parallel file system"*, ACM Transactions on Computer Systems(TOCS), 14(3), pp.225-264, Aug. 1996.

[3] Scott A. Brandt et al., *"Efficient Metadata Management in Large Distributed Storage Systems"*, Proceedings of the 11th IEEE NASA Goddard Conference on Mass Storage Systems and Technologies, Apr. 2003.

[4] G. hunt et al., *"Network Dispatcher: A Connection Router for Scalable Internet Services"*, Journal of Computer Networks and ISDN Systems, Vol.30, Elsevier Science, pp.347-357, 1998.

[5] Daniel P. Bovet et al, *Understanding the Linux Kernel*, O'Reilly and Associates, Sebastopol, 2003.

[6] http://www.mesquite.com

[7] L. mummert and M. Satyanarayanan. *"Long term distributed file reference tracing: Implementation and experience"*. Software-Practice and Experience (SPE), 26(6), pp.705-736, Jun. 1996.

[8] Kishor Shridharbhai Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, John Wiley & Sons, Inc., New York, 2002.