대용량 분산파일시스템을 위한 비공유 메타데이타 관리 기법

(A Non-Shared Metadata Management Scheme for Large Distributed File Systems)

윤 종 현 † 박 용 훈 † 이 석 재 †† 장 수 민 †††
(Jong Hyeon Yun) (Yong Hun Park) (Seok Jae Lee) (Su-Min Jang)

유 재 수 †††† 김 홍 연 ††††† 김 영 균 †††††
(Jae Soo Yoo) (Hong Yeon Kim) (Young-Kyun Kim)

요 약 최근 많은 연구가 진행 중인 대부분의 클러스터 기반 분산파일시스템은 파일에 대한 읽기, 쓰기 작업으로부터 메타데이타의 처리를 분리했다는 특징을 가지고 있다. 즉 파일시스템에 기록된 파일에 대한 권한 정보, 파일의 실제 데이타가 저장된 저장소의 위치 정보, 파일시스템의 네임스페이스 유지 등 메타데이타와 관련된 정보 및 이를 처리하는 기능을 별도의 메타데이타 서버가 관리한다. 하지만 기존 시스템의 메타데이타 관리기법들은 데이타의 분산 관리 및 입출력 성능만 중점을 두고 설계되어 있어 파일시스템 확장에 따른 메타데이타 입출력 성능 및 확장성에서 한계를 나타내고 있는 상황이다.

따라서 본 논문에서는 클러스터 기반 분산파일시스템에서 보다 나은 성능과 확장성을 제공하는 수 있는 비공유 메타데이타 관리 기법을 제안한다. 먼저 본 논문에서는 새로운 메타데이터 분할 기법으로 사전식 분할 기법을 제안한다. 다음으로 제안하는 메타데이터 분할 기법을 지원하기 위한 부하 분산 기법을 제시한다. 본 논문에서 제안하는 메타데이터 관리 기법은 기존 메타데이타 관리기법과 비교하여 확장성 및 부하 분산에서 우수함을 보인다.

키워드 : 분산파일시스템, 메타데이타 서버, 비공유 메타데이타 관리, 부하분산

Abstract Most of large-scale distributed file systems decouple a metadata operation from read and write operations for a file. In the distributed file systems, a certain server named a metadata server (MDS) maintains metadata information in file system such as access information for a file, the position of a file in the repository, the namespace of the file system, and so on. But, the existing systems used restrictive metadata management schemes, because most of the distributed file systems designed to focus on the distributed management and the input/output performance of data rather than the metadata. Therefore, in the existing systems, the metadata throughput and expandability of the metadata server are limited.

· 이 논문은 2009년 교육과학기술부(지역거점연구단육성사업/충북BIT연구중심대학육성사업단)와 교육과학기술부와 한국산업기술재단의 지역혁신인력양성사업으로 수행된 연구결과임
· 이 논문은 2008 한국컴퓨터종합학술대회에서 '대용량 분산 파일 시스템을 위한 비공유 메타데이터 서버 클러스터의 설계 및 구현'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 충북대학교 정보통신공학과
jhyun@netdb.cbnu.ac.kr
yhpark@netdb.cbnu.ac.kr
†† 정 회 원 : 한국전자통신연구원 지식이러닝연구팀
cyberdb@etri.re.kr
††† 학생회원 : 충북대학교 정보통신공학과
jsm@chungbuk.ac.kr
†††† 종신회원 : 충북대학교 전기전자컴퓨터공학부 교수
yjs@chungbuk.ac.kr
(Corresponding author임)
††††† 정 회 원 : 한국전자통신연구원 저장시스템연구팀
kimhy@etri.re.kr
kimyoung@etri.re.kr

논문접수 : 2008년 8월 25일
심사완료 : 2009년 5월 13일

정보과학회논문지: 시스템 및 이론 제36권 제4호(2009.8)

In this paper, we propose a new non-shared metadata management scheme in order to provide the high metadata throughput and scalability for a cluster of MDSs. First, we derive a dictionary partitioning scheme as a new metadata distribution technique. Then, we present a load balancing technique based on the distribution technique. It is shown through various experiments that our scheme outperforms existing metadata management schemes in terms of scalability and load balancing.

Key words : Distributed File System(DFS), Metadata Server(MDS), Non-Shared Metadata Management, Load Balancing

1. 서 론

최근 컴퓨터 및 네트워크 관련 기술의 눈부신 발전과 기하급수적인 인터넷 보급의 확대로 인해 유비쿼터스 컴퓨팅 환경으로의 변화가 매우 빠르게 진행되고 있다. 이 같은 변화는 더 많은 사람들로 하여금 인터넷상에 더 많은 정보를 생성하고 공유하도록 유도하고 있다. 유비쿼터스 컴퓨팅 환경으로의 진화는 컴퓨터시스템 관련 연구 분야에 대량의 데이타에 대한 고성능 처리와 스토리지의 무제한적인 확장 그리고 플랫폼 독립적이며 데이타의 안전한 공유를 제공할 수 있는 스토리지 시스템을 요구하고 있다. 이러한 요구를 만족시키고자 최근 대용량 클러스터기반 분산파일시스템 및 이와 관련된 기술들에 대한 다양한 연구가 활발하게 진행되고 있다.

연구가 진행중인 대부분의 클러스터기반 분산파일시스템은 파일에 대한 읽기, 쓰기 작업으로부터 메타데이타의 처리를 분리했다는 특징을 가지고 있다. 메타데이타란 데이타에 대한 구조화된 데이타로, 다른 데이타를 설명해주는 데이타다. 파일시스템의 메타데이타는 파일시스템에 기록된 파일에 대한 권한 정보, 파일의 실제 데이타가 저장된 저장소의 위치정보, 파일시스템의 네임스페이스 등과 같은 정보들로 구성되어 있다. 현재 대부분의 클러스터기반 분산파일시스템은 메타데이타와 관련된 정보와 이에 대한 처리를 별도의 메타데이타 서버(MDS : metadata server)에서 수행한다. 사용자는 파일시스템에 저장된 특정 파일에 접근하기 위해 먼저 메타데이타 서버에 접근하여 파일에 대한 권한 및 위치정보 등을 획득한 후 실제 파일의 데이타 정보를 저장하고 있는 데이타 서버에 접근할 수 있다. 이 때 메타데이타 서버에 저장되는 메타데이타 정보의 크기는 수십 Byte ~ 수 KB 정도로 실제 파일의 데이타와 비교해 상대적으로 매우 작다. 따라서 전체 파일시스템에서 메타데이타가 차지하는 저장소 크기는 실제 데이타에 비해 무시해도 좋을 정도로 작기 때문에 일반적으로 메타데이타 관리기법에 대한 연구는 거의 진행되지 않고 있다.

그러나 데이타 크기가 아닌 I/O 연산의 관점에서 볼 때, 메타데이타의 처리를 분리하여 관리하는 것은 매우 많은 작업 부하를 발생시킨다. 실제 파일시스템의 디렉토리나 파일의 데이타를 접근하여 읽기, 쓰기, 수정 등의 작업을 처리하기 위해서는 반드시 그 파일의 메타데이타에 대한 접근이 먼저 처리되어야 한다. 따라서 실제 파일에 대한 I/O 연산은 반드시 메타데이타에 대한 I/O 연산을 함께 발생시킨다. 또한 디렉토리 탐색, 권한, 소유자 변경과 같이 실제 파일데이타를 직접 접근할 필요가 없는 연산의 경우에도 메타데이타에 대한 I/O 연산이 발생하기 때문에 메타데이타 I/O 요청은 실제 파일데이타 연산과 비교하여 매우 빈번하게 발생하게 된다.

고성능 컴퓨터에 대한 벤치마크 기관인 SPEC의 연구결과에 따르면 클러스터 파일시스템에 대한 I/O연산의 분석 결과, 전체 I/O 연산 중 60% 이상이 메타데이타와 관련된 I/O 연산이었음을 알 수 있다[1]. 또한 메타데이타 서버에 대한 작업 부하는 사용자의 응용에 따라 그 정도가 시시각각 매우 크게 변화하는 특성을 갖는다[2]. 따라서 위의 여러 연구 결과들을 종합, 분석해 보면 메타데이타에 대한 I/O 처리 성능이 전체 파일시스템 성능에 매우 큰 영향을 미친다는 것을 확인할 수 있다.

또한 그 동안 크게 고려하지 않았던 메타데이타의 크기 부분에서도 여러 문제점들이 발생하고 있다. Google과 같은 검색 서비스의 경우, 관리할 데이타 파일의 개수가 80억 개를 넘어서고 있으며, UCC 서비스를 제공하고 있는 몇몇 대형 포털 서비스들의 경우, 사용자들이 하루에 등록하는 콘텐츠의 종류와 양이 기하급수적으로 늘어나고 있다. 콘텐츠 증가에 따라 클러스터 파일시스템 환경에서 파일의 실제 데이타를 저장하는데 사용되는 시스템은 쉽게 확장이 가능하지만, 메타데이타를 관리하는 메타데이타 서버는 대부분 확장을 고려한 설계가 적용되지 않았다. 때문에 메타데이타 서버의 확장이 거의 불가능하거나, 확장하는데 많은 비용이 필요하다.

현재 상용화되거나 연구 중인 대부분의 클러스터기반 분산파일시스템은 효과적인 메타데이타 관리기법 부재로 전체 시스템의 성능 및 그 확장이 크게 제약 받고 있다. 따라서 클러스터기반 분산파일시스템의 특성을 정확히 파악하고 이를 잘 활용해 분산파일시스템 환경에서 보다 나은 성능과 확장을 제공할 수 있는 메타데이타 관리기법과 관련 기술의 연구가 절실히 요구되고 있다.

따라서 본 논문에서는 성능과 확장성이 뛰어나며, 안정적인 메타데이타 서비스를 제공하는 비공유 메타데이타 관리기법을 설계하고, 설계된 관리기법을 이용하여

메타데이타 서버 클러스터를 구현한다. 구현된 메타데이타 서버 클러스터를 통해 제안하는 메타데이타 관리기법과 기존 메타데이타 관리기법을 비교, 평가하여 설계한 기법의 우수성을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 메타데이타 관리기법을 비교, 분석한다. 3장에서는 제안하는 메타데이타 관리기법을 메타데이타 분할 기법 및 메타데이타 부하분산기법 중심으로 설계한다. 4장에서는 설계된 관리기법을 이용하여 구현된 메타데이타 클러스터 서버의 성능을 기존 메타데이타 관리기법을 적용한 메타데이타 클러스터 서버와 비교, 분석하여 제안하는 기법의 우수성을 보인다. 5장에서는 결론을 맺고 향후 연구 방향을 정리한다.

2. 관련연구

분산파일시스템은 단일 서버가 아닌 여러 대의 서버에 파일시스템 데이타를 분산시켜 성능을 향상시키고 확장성을 높이는 것을 목적으로 한다. 대표적인 분산파일시스템으로 NFS, AFS, GFS, Coda 등이 있다. 그러나 대부분의 분산파일시스템들은 파일의 데이타와 메타데이타의 관리를 분리하지 않고 같은 로컬 디스크 상에 저장해 관리하는 구조를 갖고 있다. 이런 구조의 분산파일시스템들은 확장성과 I/O 성능이 제한적인 문제점을 갖고 있다.

최근 많은 연구가 진행되고 있는 클러스터 파일시스템에서는 그림 1과 같이 파일데이타와 메타데이타의 저장위치를 완전히 분리함으로써 관리 효율을 높이고 I/O 성능을 개선하기 위한 노력들을 계속하고 있다[3]. 실제 파일의 데이타 I/O연산과 메타데이타 I/O연산을 분리해 시스템의 성능을 높이려는 시도는 예전부터 연구되어 왔다[4-9]. 이렇게 분리된 메타데이타 정보를 관리하는 메타데이타 관리기법은 메타데이타 저장 관리 형태에 따라 공유기반 메타데이타 관리기법과 비공유기반 메타데이타 관리기법으로 나누어 볼 수 있다[10].

공유기반 메타데이타 관리기법은 모든 메타데이타 서버가 동일한 메타데이타 정보, 즉 복제 정보를 유지하는 것이다. 이는 각 서버가 다른 서버의 도움 없이 언제든지 전체 파일시스템의 메타데이타에 대한 접근을 가능하게 한다. 따라서 특정 메타데이타에 대한 읽기 연산이 매우 빈번하게 발생할 경우 여러 서버를 통해 동시에 처리하는 것이 가능하기 때문에, 읽기 연산에서는 높은 부하분산 효과를 갖는다. 그러나 쓰기 연산이 발생하는 경우 동시성 제어를 위해 모든 서버 간에 정보교환이 필요하다. 또한 각 메타데이타 서버가 모든 파일시스템의 메타데이타를 유지하기 때문에 메타데이타를 저장하는 저장공간을 효과적으로 사용할 수 없으며, 지속적인 메타데이타 증가에 따른 저장소의 추가 역시 한계가 있다.

비공유기반 메타데이타 관리기법은 분산된 각 서버가 파일시스템의 메타데이타를 적절한 메타데이타 분할기법을 사용하여 일부분씩 나누어 관리하는 기법이다. 기본적으로 각 서버는 자신이 관리하는 메타데이타 영역에 대한 I/O 요청만 처리한다. 관리영역 외의 다른 메타데이타 영역에 대한 I/O 요청은 처리하지 않거나, 해당 영역을 담당하는 서버의 도움을 받아야 처리가 가능하다.

표 1은 대표적인 비공유 메타데이타 관리 기법의 특징 및 장, 단점을 요약한다[10-12]. 비공유 메타데이타 관리 기법들을 정리해보면 크게 해시 기법과 서브트리 기법으로 분류할 수 있다.

서브트리 기법은 메타데이타 분할 시, 파일시스템의 네임스페이스 구조를 고려하여 메타데이타를 분할한다. 즉 동일한 부모 디렉토리를 갖는 서브 디렉토리 및 파일들은 동일한 메타데이타 서버에 할당된다. 이는 같은 디렉토리에 있는 다른 파일의 메타데이타에 추가적인 접근이 필요한 경우, 참조지역성에 의해 검색 비용을 절약할 수 있다.

서브트리 기법은 크게 NFS 등의 시스템에서 사용하

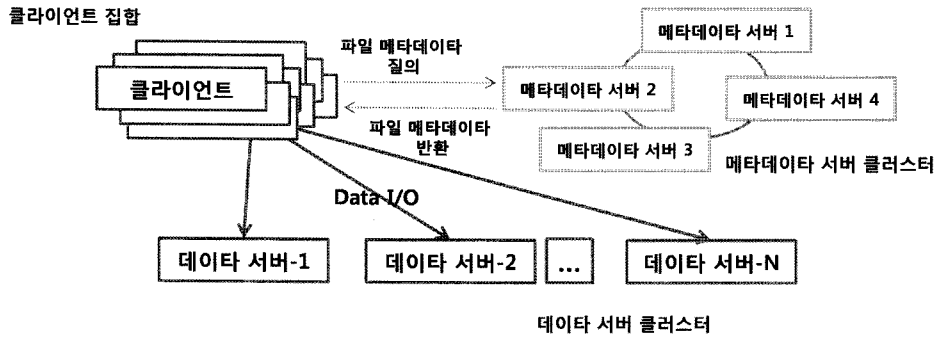


그림 1 독립된 메타데이타 처리 서버를 갖는 분산파일시스템

표 1 대표적인 비공유 메타데이타 관리 기법

방식	특징	장점	단점
정적트리 분할	•메타데이타를 디렉토리 계층(namespace)에 따라 정적으로 분할 Ex) NFS, AFS, Coda	• 디렉토리 계층정보 유지 • 참조 지역성 유지	•최초 분할에 따라 성능 좌우 •특정 디렉토리에 연산 집중 시 취약 •메타데이터 수 증가에 따른 서버 확장에 제약
해시 (Hash)	•메타데이타를 관리하는 MDS의 위치를 해시 함수로 결정 Ex)Lustre, RAMA, Vesta	•각 서버간 동등한 메타데이타 정보 유지 •이론적으로 균등한 부하 분산 가능	•메타데이타의 변경 연산에 취약 •서버 구조 변경에 제약
Lazy Hybrid (LH)	•해시와 서브트리 분할 특성을 결합 •MLT(메타데이타 검색 테이블)와 ACL(접근 제어 리스트)를 사용	•접근제어리스트를 유지하여 추가적인 디렉토리 순회 비용 절감 •서버 클러스터 구성 변경 시 Lazy Update로 연산 비용 절감	•메타데이타의 변경 연산에 취약 •권한 정보 변경 시 ACL 리스트의 갱신 요구
동적 서브트리 분할 (DSP)	•메타데이타를 디렉토리계층 구조에 따라 분할 •부하에 따라 분할된 디렉토리계층을 동적으로 분할, 분산할 수 있음 •(디렉토리 이동 & 해시 기법 적용)	•디렉토리계층 정보를 유지하여 추가적인 디렉토리 순회 비용 절감 •동적 작업 부하를 해결하기 위한 복제 기법과 디렉토리 재분할 기법 적용	•메타데이타 연산을 위한 서버간 메시지 포워딩 발생 •동적 디렉토리 분할 시 분할 기준이 불명확 •분산된 디렉토리 계층 정보의 재조정 요구

는 정적 서브트리 기법과 CEPH에서 사용하는 동적 서브트리 기법으로 분류된다[12]. 정적 서브트리는 파일시스템의 크기가 그리 크지 않은 경우 비교적 관리가 용이하지만, 메타데이타의 분할이 초기 관리자의 개입에 의해 결정되기 때문에 본질적으로 부하 분산이 불균등하기 이루어진다.

동적 서브트리 분할 기법은 초기 분할은 정적 서브트리 분할 기법과 유사하다. 그러나 각 메타데이타 서버의 부하가 일정 수준에 도달하면 동적으로 분할된 메타데이타 정보를 재배치하고 이에 따라 해당되는 메타데이타를 적절한 메타데이타 서버로 분산시킨다.

동적 서브트리는 정적 서브트리와 대비하여 시스템 운영 시, 발생되는 부하에 효과적으로 대처할 수 있다. 그러나 시스템이 운영되면서 메타데이타 분할 정보가 복잡해지고, 따라서 시스템에 접근하는 클라이언트들에게 정확한 메타데이타 분할 정보를 알려주지 않는다. 따라서 불필요한 메타데이타 서버의 접근 비용이 발생할 수 있다. 또한 복잡한 부하 분산 방법으로 부하 분산을 효과적으로 관리하기 어려운 단점이 있다.

해시 기반 기법은 메타데이타 분할 시, 모든 메타데이타 서버에 균등하게 메타데이타를 분할하기 위한 목적으로 제안되었다. 따라서 파일시스템의 네임스페이스 구조는 무시되며, 오직 특정 해시 함수에 의해 메타데이타를 관리하는 메타데이타 서버의 위치가 결정된다. 따라서 모든 메타데이타 서버가 언제나 균등한 메타데이타를 보유하게 된다. 이는 잠재적으로 모든 서버에게 균등한 부하를 분산한 것과 동일하다[10,11].

해시 기법은 크게 일반적인 해시 함수를 적용한 Naive Hash와 해시와 정적 서브트리 분할 방법의 장점을 결합한 Lazy Hybrid로 분류된다. Naive Hash는 일반적인 해시 함수를 적용한 기법이다. 메타데이타의 파일시스템 네임스페이스 정보를 해시 함수에 입력값으로 사용하여 그 결과에 따라 메타데이타를 관리할 서버가 결정되는 방식이다.

Naive Hash는 클러스터를 구성하는 각 메타데이타 서버에 균등하게 메타데이타를 분배하고 관리할 수 있다. 그러나 파일시스템의 네임스페이스를 무시한 메타데이타 분배로 인해 디렉토리 혹은 파일 검색 시, 최악의 경우 모든 메타데이타 서버를 접근해야 하는 단점이 있다. 또한 메타데이타 서버의 구조 변경에 취약한 단점이 있다.

이와 같은 Naive Hash의 단점을 극복하기 위해 Lazy Hybrid가 제시되었다[11]. Lazy Hybrid는 접근 제어 리스트(ACL: Access Control List)이라는 특수한 접근 제어 정보를 이용하여 Naive Hash의 단점을 극복했다. 또한 메타데이타 서버의 구조 변경 시, 지연처리를 통해 시스템의 부하를 감소시키고 있다. 그러나 ACL을 작성하기 위해 필요한 비용 및 지연 처리로 인한 응답성 저하와 같은 문제점을 가지고 있다.

3. 제안하는 메타데이타 관리기법

이 장에서는 기존 메타데이타 관리기법들이 갖고 있는 느린 파일 탐색 성능 문제를 개선하고, 서버 간 균형 유지 및 서버의 추가/제거에 유연하게 대처할 수 있는 새로운 분할기법을 제안한다. 또한 제안하는 분할기법의 특성을 고려하여 특정 메타데이타 서버에서 작업 부하 발생 시, 해당 부하를 빠르게 분산시킬 수 있는 부하분산기법을 제안한다.

3.1 제안하는 메타데이타 분할 기법

본 논문에서는 기존 메타데이타 관리 기법의 장점을 결합하여 새로운 메타데이타 분할 기법을 제안한다. 즉 파일시스템 네임스페이스의 계층적 구조를 보호하고, 초

기 각 메타데이타 서버에 균등하게 메타데이타를 분배할 수 있는 사전식 분할 기법(DicPS : Dictionary Partitioning Scheme)을 제안한다.

제안하는 메타데이타 분할 기법은 시스템의 메타데이타를 사전식으로 정렬한다. 즉 메타데이타의 경로명에 따라 메타데이타를 정렬한다. 정렬이 이루어지면 계층적 구조로 이루어진 시스템의 메타데이타를 선형적 구조로 표현할 수 있다. 그림 2는 계층적 구조로 표현된 디렉토리 구조를 사전식으로 정렬하고 선형적 구조 표현으로 바꾸어 나타낸 것이다. 정렬 기준은 메타데이타의 네임스페이스 정보를 사용한다. 즉 해당 메타데이타에 접근하기 위해 필요한 전체 메타데이타 경로를 하나의 단어로 인식하고, 이를 사전식 정렬을 통해 정렬한다. 따라서 그림 2와 같이 "/a/a"는 "/a/b" 보다 우선 정렬되며, "/a"는 "/a/a"와 같은 경로에서 시작하지만 그 길이가 작기 때문에 먼저 정렬된다. 또한 정렬 기준의 일관성을 위해 제안하는 기법은 디렉토리 내의 파일 메타데이타 정보를 우선 정렬한 후, 하위 디렉토리에 대해 정렬 작업을 수행한다.

이렇게 선형적 구조로 변환 및 정렬이 완료되면 서버 수와 전체 메타데이타의 수를 고려해, 각 그룹에 포함된 디렉토리 및 파일 메타데이타의 개수가 유사해지는 위치를 지정하여 클러스터 내의 서버 수만큼의 그룹으로 나눈다. 이렇게 나눈 그룹은 클러스터내의 각 서버에 할당해 관리하도록 한다. 그림 3은 선형적 구조로 표현된 디렉토리를 유사한 개수의 파일을 관리하는 그룹으로 분할하여 각 서버에 할당한 예이다.

메타데이타를 분할하면 메타데이타 분할 기준 정보를 생성할 수 있다. 분할 기준 정보란 각 메타데이타 서버가 관리하는 메타데이타의 범위를 의미한다. 즉 그림 3의 MDS A는 "/"부터 "/b" 디렉토리의 하위 파일 메타데이타(서브 디렉토리 제외 - MDS B에서 관리)까지의 메타데이타를 관리한다. 따라서 MDS A의 분할 기준 정보는 "/"부터 "/b"까지이다. 이렇게 생성된 분할 기준 정보 역시 메타데이타 정보 정렬 순서에 따라 오름차순으로 정렬되어 관리된다.

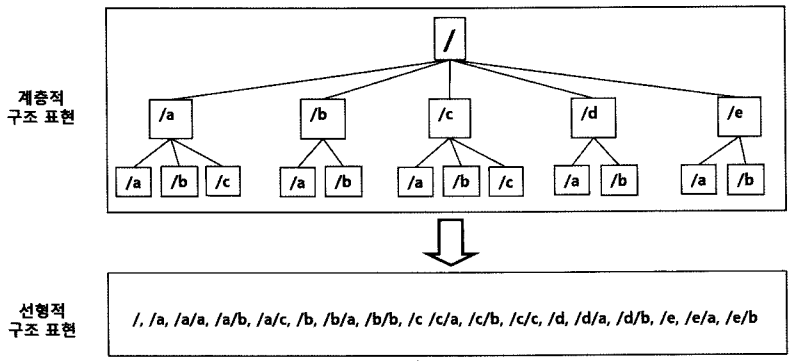


그림 2 제안하는 메타데이타 구조 표현 기법

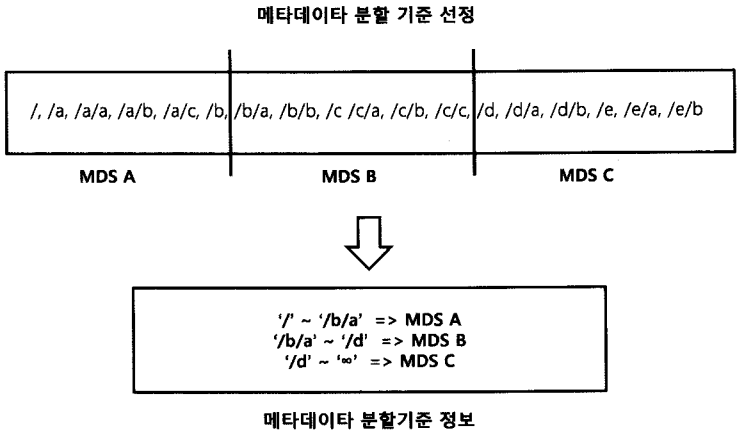


그림 3 메타데이타 분할지점 선택 및 분할기준정보 작성

메타데이타 분할 기준 정보에는 버전 정보가 할당된다. 버전 정보는 메타데이타 분할 기준이 변경되는 경우 갱신된다. 분할 기준 정보는 이 버전 정보와 함께 모든 메타데이타 서버에서 동일하게 관리한다. 클라이언트의 경우에도 한 번 메타데이타 서버에 접근하면 이 정보를 가져가 캐시에 저장하고 사용한다. 클라이언트는 캐시의 분할 기준 정보를 이용하여 서버에 접근한다. 이때 캐시된 분할 기준 정보의 버전과 서버에 등록된 분할 기준 정보의 버전 정보를 비교, 확인한다. 만약 클라이언트와 서버의 메타데이타 분할 기준 정보의 버전 정보가 일치하지 않으면 클라이언트가 캐시한 메타데이타 분할 정보는 더 이상 유효하지 않다 따라서 분할 기준 정보를 서버로부터 새로 받아 간다.

메타데이타 클러스터를 관리하는 관리자는 특정한 경계값을 부여하여 메타데이타 정보를 각 메타데이타 서버에 불균등하게 분할할 수도 있다. 이는 특정 메타데이타의 경우 빈번한 접근이 발생하지만, 대부분의 메타데이타는 거의 접근이 발생하지 않기 때문이다.

예를 들어 "/etc/passwd"나 "/temp"와 같이 중요한 시스템 파일 혹은 임시파일은 매번 빈번하게 접근되는 반면, /home의 데이타는 사용자의 요청에 따라 상대적으로 덜 빈번하게 접근된다. 따라서 메타데이타 서버에 메타데이타를 분할할 때 자주 접근되는 디렉토리를 포함하는 메타데이타 서버에는 0~1 사이의 경계값을 부여하여 한 서버가 관리할 수 있는 메타데이타 수를 특정한 최소, 최대값을 설정할 수 있다.

그림 4는 이러한 경계값을 부여하여 제안하는 메타데이타 분할 기법을 적용한 예이다. 즉 "/"에서 "/A/D/Q" 경로 사이에 있는 "/A/C/" 하위의 파일들이 자주 접근된다고 가정하자. 이 경우 MDS001에 사용자가 임의의 가중치를 부여하여 일정 수 이하의 메타데이타만 유지하도록 한다. 즉 그림 4의 경우 "/A/C/"에 대한 가중치 0.1을 부여하여 MDS001 은 (전체 파일시스템 메타데이타 수/총 메타데이타 서버 수) * (1 - 가중치)개 미만의 메타데이타만 유지하게 한다.

제안하는 메타데이타 분할 기법의 메타데이타 탐색 기법은 분할 기준 정보를 사용하여 처리된다. 분할 기준 정보를 갖고 있는 클라이언트는 파일 또는 디렉토리 탐색 시 분할 기준 정보를 통해 요청을 보낼 서버를 선택한다. 파일시스템에 처음 접속한 사용자는 분할 기준 정보를 갖고 있지 않기 때문에 찾으려는 정보가 어느 서버에 있는지 알 수 없다. 따라서 처음 접속한 사용자는 먼저 메타데이타 서버 클러스터를 구성하는 임의의 서버에 임의로 접근하여 필요한 분할기준정보를 요청한다.

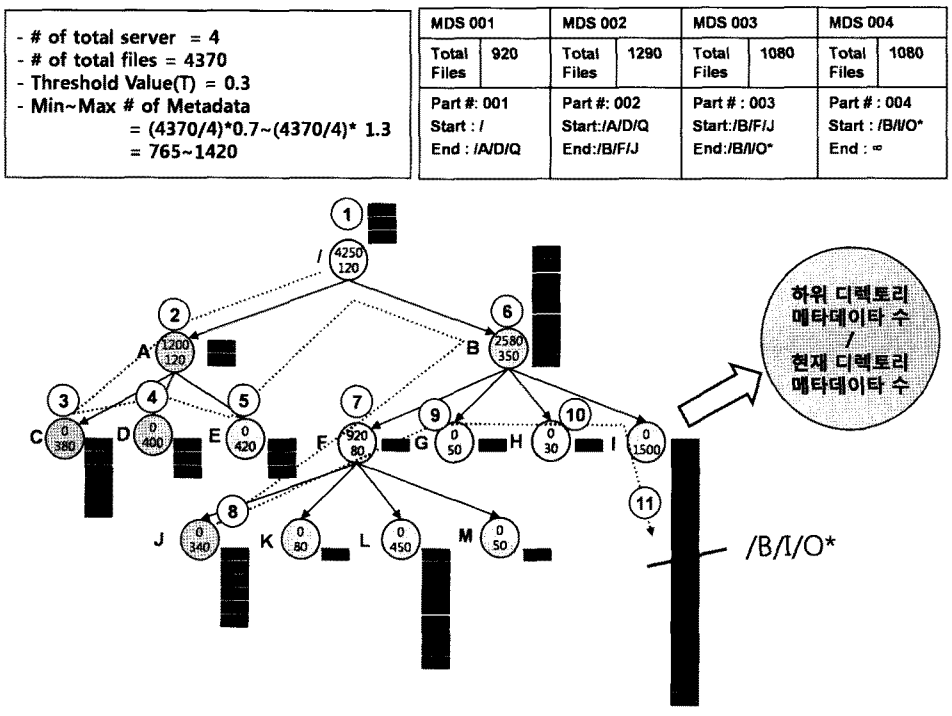


그림 4 제안하는 메타데이타 분할 기법의 적용 예시

요청을 받은 서버는 요청된 메타데이타 경로를 분할기준정보와 비교하여 해당 디렉토리 또는 파일의 메타데이타를 관리하는 서버를 알아낸다. 처음 요청을 받은 서버는 요청한 메타데이타가 다른 서버에 있으면 클라이언트의 요청을 해당 서버로 전달한다. 그림 5는 클라이언트의 탐색 요청을 처리하지 못해 다른 서버로 요청을 전달하는 과정을 나타낸다. 처음 해당 파일시스템에 접근하는 클라이언트는 MDS A부터 C까지의 서버 중, 임의의 한 서버에 접근한다. 임의의 서버에 접근하여 메타데이타 분할 기준 정보를 얻으면 요청하는 파일의 메타데이타 정보가 어떤 메타데이타 서버의 메타데이타 분할 기준 정보에 포함되는지에 따라 요청을 처리할 수 있는 메타데이타 서버를 확인할 수 있다. 그림 5에서는 "/a/z/text.txt"는 메타데이타 분할 기준 정보에 따라 현재 MDS A에서 관리하고 있다. 따라서 MDS C는 해당 정보를 클라이언트 대신 MDS A에 문의하고, 해당 파일의 메타데이타 정보를 메타데이타 분할 기준 정보와 함께 클라이언트에게 반환한다.

클라이언트가 이미 메타데이타 서버에 접근했었던 경우에는 메타데이타 분할 정보를 이용해 접근하려는 메타데이타를 관리하는 서버에 바로 요청을 보낸다. 클라이언트가 요청한 메타데이타 정보가 해당 서버에 존재하면 서버는 요청을 바로 처리한다. 서버에 요청한 메타데이타 정보가 없을 경우 클라이언트의 요청을 해당 요청을 처리할 수 있는 서버로 전달하고, 서버가 관리 중인 분할 정보의 버전과 클라이언트에 캐시된 버전을 비교해 클라이언트의 정보를 최신의 것으로 갱신한다.

메타데이타의 추가 및 삭제 과정 역시 탐색과 동일한 절차를 거쳐 수행된다. 먼저 추가하려는 파일 또는 디렉토리 경로가 어떤 서버에 의해 관리되어야 하는지를 분할 기준 정보를 이용해 찾는다. 탐색과 마찬가지로 처음 접속하는 클라이언트인 경우 서버에서 요청을 처리할 수 있는 서버를 찾아 요청을 전달하고 클라이언트에게 메타데이타 분할 정보를 전송한다.

3.2 제안하는 비공유 메타데이타 부하 분산 기법

별도의 메타데이타 서버를 갖는 클러스터 파일시스템의 작업 부하는 일시적인 부하와 지속적인 부하로 구분할 수 있다. 일시적인 작업 부하는 특정 메타데이타에 대한 클라이언트들의 요청 집중이다. 예를 들어 인기 있는 UCC 동영상의 경우 일시적으로 매우 빈번한 접근 요청이 발생할 수 있다. 이러한 작업 부하는 일시적으로 지속되며 대부분 특정 시간 이후에는 더 이상 발생하지 않는다. 지속적인 부하는 한 서버에 일정 수준 이상의 메타데이타가 집중되어 저장되는 경우 발생한다. 즉 한 디렉토리에 수천~수만 개의 파일이 저장되거나 빈번한 메타데이타의 이동이 발생하는 경우 해당 서버는 지속적인 작업 부하에 노출된다. 이러한 상황들은 장기적으로 메타데이타 서버들 간의 메타데이타 수의 불균형을 발생시켜 지속적인 작업 부하를 발생시킨다. 따라서 이들 부하를 효과적으로 해결할 수 있는 메타데이타 관리기법이 요구된다[13]. 본 절에서는 이들 작업 부하를 해결하는 기법과 함께 서버 구조 변경 시의 처리 기법을 기술한다.

3.2.1 일시적인 부하 처리

비공유 메타데이타 관리기법은 클러스터를 구성하는

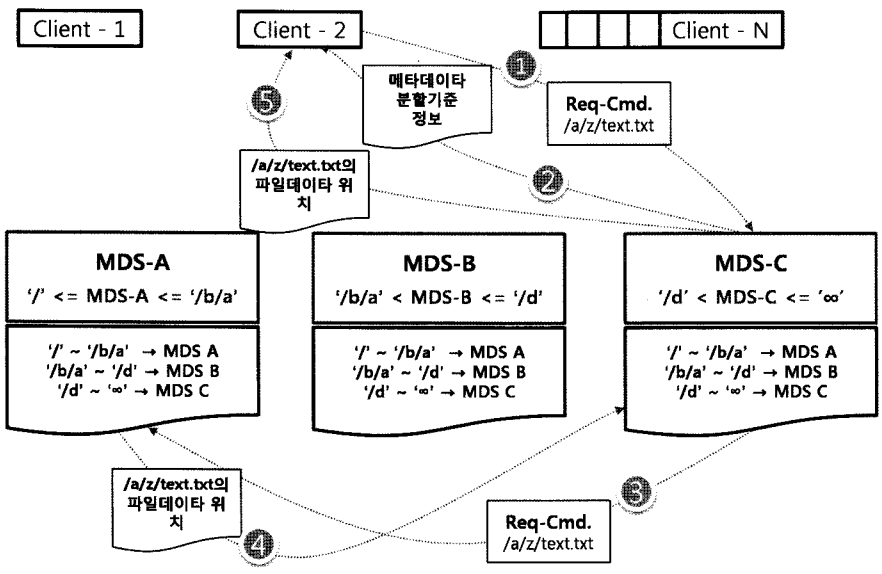


그림 5 메타데이타 검색 예제

각 메타데이타 서버에 메타데이타를 분할하여 할당한다. 따라서 분할된 각 메타데이타들에 대해서는 그 메타데이타들이 저장된 메타데이타 서버가 관리한다. 만약 특정 메타데이타에 대해 접근하기 위해서는 그 메타데이타를 관리하는 서버에 접근해야 한다.

실제 이러한 파일시스템을 사용하는 응용에서는 특정 파일에 대한 접근이 빈번하게 발생하게 된다. UCC나 웹 서버의 특정 웹 페이지와 같이 클라이언트들이 일시적으로 빈번하게 접근하는 데이타 파일이 존재한다. 이러한 일시적인 접근이 빈번하게 일어나는 경우, 해당 파일 데이타의 메타데이타 역시 빈번하게 요청된다. 따라서 이러한 작업 부하는 해당 메타데이타를 관리하는 서버에서도 동일하게 발생하게 된다.

이를 해결하기 위해 제안하는 메타데이타 관리기법은 복제 캐시를 사용한다. 복제 캐시는 빈번하게 요청되는 메타데이타의 정보를 복제하여 캐시한다. 그림 6은 이러한 복제 캐시가 사용되는 것을 설명한다. 먼저 각 서버는 클라이언트들의 작업 요청을 작업 큐로 관리한다. 작업 큐에 저장된 작업 요청은 지정된 시간 주기로 계산된다. 만약 지정된 시간 주기 동안 요청된 작업의 대다수가 특정 메타데이타에 대한 작업 요청으로 구성되면 해당 서버는 그 메타데이타에 대한 복제 작업을 다른 서버에게 요청할 수 있다. 해당 서버는 메타데이타에 대한 복제 요청을 수신한 다른 서버들 중에서 가장 빠른 응답을 보인 서버를 선택하여 해당 메타데이타에 대한 복제 작업을 요청한다. 복제 작업이 종료되면 클라이언트들에게 해당 메타데이타 정보가 다른 서버에 복제되어 있음을 피드백한다. 만약 해당 메타데이타에 대한 작업 부하가 더 이상 발생하지 않는 경우 해당 서버는 자신이 복재한 메타데이타 정보를 회수할 수 있다.

3.2.2 지속적인 부하 처리

시스템의 각 메타데이타 서버가 보유한 메타데이타 수는 초기 분할 과정을 통해 균등하게 배분된다. 하지만 시스템의 운영에 따라 메타데이타의 수는 증감을 반복하게 된다. 특히 한 디렉토리에 수천~수만 개의 파일이 저장되거나 디렉토리 메타데이타의 이동이 발생하는 경우 각 서버의 메타데이타의 수는 장기적으로 불균등해질 수 있다. 각 서버들이 보유한 메타데이타 수의 불균등이 지속될 경우 특정 서버에게 좀더 많은 작업 부하가 부여될 수 있다. 따라서 서버들이 보유한 메타데이타의 수를 균등하게 유지할 수 있는 기법이 요구된다.

시스템의 장시간 운영으로 각 서버가 관리하는 메타데이타의 개수 편차가 일정 수준 이상으로 커져 서버 간 균형이 잘 맞지 않는 상황이 발생하면 전체 시스템에 대한 메타데이타 분할 기준점 재조정 작업이 고려될 수 있다. 재조정이 필요한 시점에 대해서는 다양한 기준이 있을 수 있다. 일반적으로 데이타베이스 시스템의 경우 데이타베이스 파일의 확장을 결정하는 시점은 현재 설정된 파일 용량의 80% 수준에 해당하는 만큼 데이타가 축적되면 처음 생성한 파일 크기의 5~20% 정도 범위에서 파일 크기를 지속적으로 확장하는 정책을 사용하고 있다. 메타데이타 서버 클러스터의 경우 메타데이타 자체의 크기가 작고, 차지하는 용량은 메타데이타의 개수에 비례하기 때문에 개수를 기준으로 운영자가 임계치를 설정하는 방식으로 관리하는 것이 적절할 것이다. 제안하고 있는 메타데이타 관리기법에서는 분할을 결정하는 기준 자체가 메타데이타의 개수 균형이기 때문에 재조정이 필요한 시점에 대한 평가 역시 일정한 임계치 이상 서버 간 개수 편차가 벌어지는 시점을 선택한다.

분할 기준의 재조정은 전체 메타데이타에 대한 개수 및 분포 파악이 필요하고 거의 모든 디렉토리 메타데이타를 읽어야하기 때문에 자주 수행하는 것은 시스템의 정상적인 운영에 영향을 미칠 수 있다. 따라서 시스템

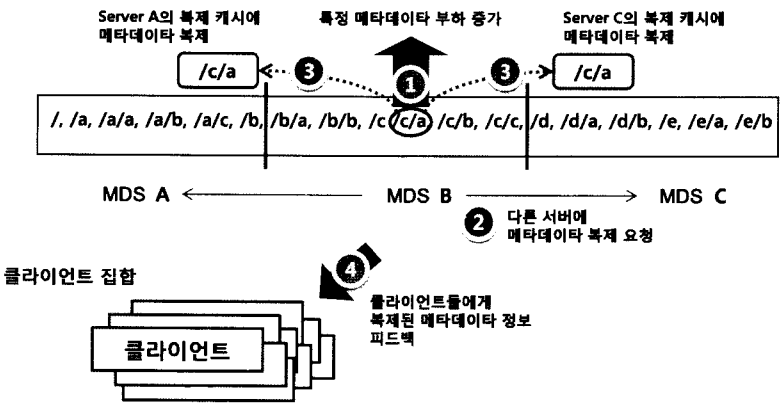


그림 6 특정 메타데이타의 일시적인 부하 처리

관리자는 급격한 변동으로 인해 재조정이 필요한 상황이 아니라면 데이타의 증가 추세와 클라이언트들의 I/O 요청 패턴을 고려하여 재조정을 하도록 분할 기준 변경 판단을 위한 메타데이타 변화량에 대한 임계치를 설정해 줄 필요가 있다.

그림 7은 MDS B에 지속적으로 메타데이타가 증가하여 재조정이 수행되는 경우 처리과정을 보여준다. 먼저 MDS B는 이웃 서버들과 균형차를 계산한다. MDS B는 주변 MDS(이전 및 다음 분할 위치를 관리하는)들과 정보를 교환하고 새로운 분할 기준점을 선정한다. 새로 선정된 기준점에 따라 MDS B로부터 MDS A와 MDS C로 이동할 메타데이타들이 발생하면 4.2.3의 '메타데이타의 이동 부하 처리'에서와 같은 과정을 거쳐 이동 처리한다.

3.2.3 메타데이타의 이동 부하 처리

제안하는 메타데이타 분할 기법은 디렉토리의 이름을 기준으로 분할 기준점을 설정한다. 따라서 클라이언트 사용자가 디렉토리 이름을 변경할 경우 변경된 이름에 해당하는 메타데이타를 관리하는 서버가 바뀔 수 있다. 이런 경우 변경된 디렉토리와 하위 디렉토리에 대한 메타데이타를 모두 변경된 디렉토리를 관리하는 서버로 이전해 줘야 한다. 하위 디렉토리에 포함되는 메타데이타의 수가 많을 경우 이전에 매우 큰 비용과 시간이 소모된다. 이름이 변경되거나 다른 위치로 이동한 디렉토리가 루트에 가까운 상위레벨의 디렉토리일 경우 이런 문제가 더 크게 발생할 수 있다. 제안하는 기법에서는 변경을 바로 처리하지 않고 시스템의 유휴시간에 이전 작업을 하도록 스케줄을 설정하고, 해당 디렉토리에 잠금(lock)을 설정한다. 분할 정보에는 이전해야 할 대상 위치에 대한 임시적인 분할 정보를 추가하고 실제 요청 처리는 변경 전 서버에서 이전 작업이 완료될 때까지 계속 처리하도록 한다. 메타데이타의 이전 작업은 원본 서버의 메타데이타를 모두 대상 서버로 복사한 후, 원본 서버의 해당 디렉토리 레코드를 모두 삭제 표시하고 잠금을 해제하면 종료된다. 그림 8은 '/c'를 '/e/c'위치로 이동한 경우 처리 과정과 임시로 생성된 분할 기준 정보 테이블을 보여준다.

3.2.4 메타데이타 서버의 추가 및 삭제

메타데이타 서버 클러스터 시스템의 전체적인 사용량이 한계에 가까워지면 서버를 신규로 추가해 증설해야 한다. 반대로 처음 시스템 구축시 보다 관리할 데이타의 양이 줄어들거나 서버를 교체해야 할 경우 서버를 제거해야 한다. 서버의 추가 및 제거는 모두 서비스가 제공되는 온라인상에서 처리 될 수 있어야 한다.

본 논문에서 제안하는 기법에서는 서버의 추가 시 현재 사용 중인 서버들 중 가장 부하(요청 수 또는 메타데이타의 개수)가 큰 서버가 관리하는 영역의 일부를

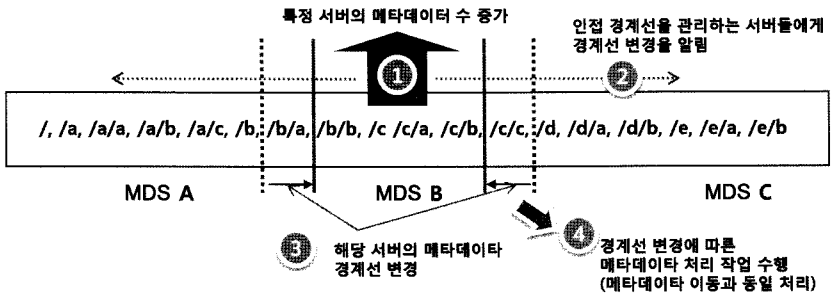


그림 7 메타데이타 수 증가에 따른 분할 정보 재조정

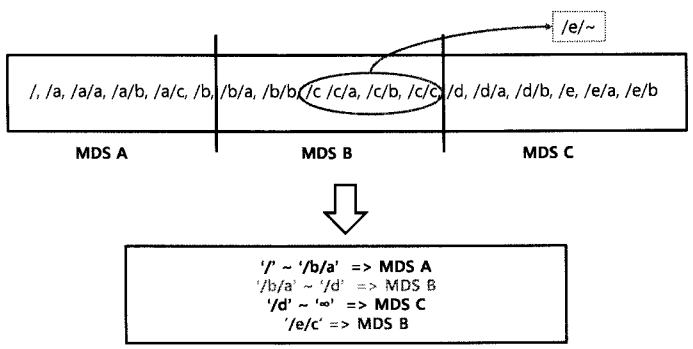


그림 8 디렉토리 메타데이타 이동에 따른 처리('/c'를 '/e/c'로 이동)

분할해 신규 서버에 할당하는 기법을 기본으로 사용한다. 하지만 시스템 관리자가 임의로 할당되지 않은 범위를 추가해 할당하거나 두 대의 서버에서 영역을 각각 분할해 재조정하는 것도 가능하다. 두 대 이상의 서버에서 분할영역을 가져오는 것은 하나 이상의 서버가 관리하는 메타데이타 전부를 이전해야 하기 때문에 이전비용이 매우 커져, 전체 클러스터 시스템을 새로운 다른 클러스터 시스템으로 완전 이전하는 것과 같은 경우를 제외하고 실효성이 없다. 그림 9와 10은 메타데이타 서버 추가 및 삭제 시의 과정을 설명한다.

클러스터를 구성하고 있던 서버 중 하나를 제거하는 경우에는 제거 대상 서버가 관리하던 메타데이타의 개수를 절반에 가깝게 나눌 수 있는 분할 기준점을 먼저 선정한다. 선정한 분할 기준점을 중심으로 이전 영역과 다음 영역을 관리하는 서버들에게 메타데이타의 이전이 모두 끝나면 서버를 제거할 수 있다. 이때 제거되는 서버의 메타데이타 개수가 많은 상황이라면 이전을 받은 서버들이 관리 한계용량을 초과하는 경우가 발생할 수 있다. 서버의 수가 더 많은 상황이라면 연쇄적인 분할 기준점 재조정 및 이전이 발생할 수 있다.

4. 실험 및 성능 평가

4.1 실험 환경

성능 평가를 위해 사용한 시스템 환경은 표 2와 같다. 실험은Redhat Linux Fedora core6 를 설치한 8대의 시스템에서 메타데이타 서버를 운영하였다. 각 MDS 시스템은 RAM 2G, 1 HDD SAS 74G를 갖는다. 메타데이타 서버에 메타데이타를 요청하는 Client는 RAM 2G, Linux kernel v2.6.10의 사양을 갖추고 있으며 Gigabit Ethernet을 통해 서로 연결되어 있다.

각 메타데이타 서버는 메타데이타 정보를 저장하고 색인하기 위해 BekeleyDB 4.6 버전을 사용한다. 메타데이타 정보는 색인 정보와 실제 데이터로 구성되며, 색인은 Bekeley DB에서 제공하는 B-Tree를 사용한다. 데

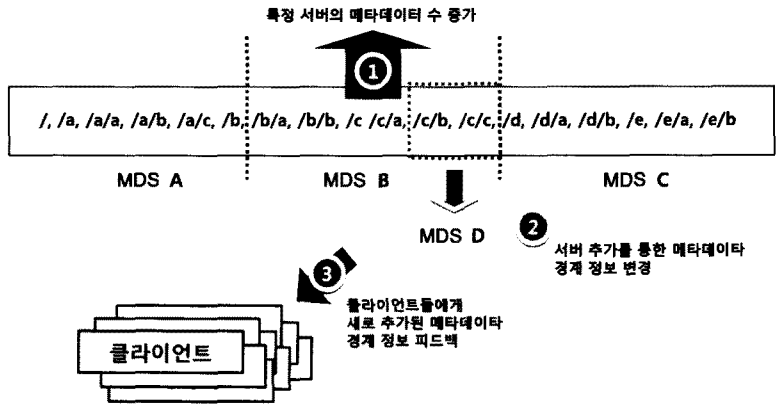


그림 9 메타데이타 서버의 추가

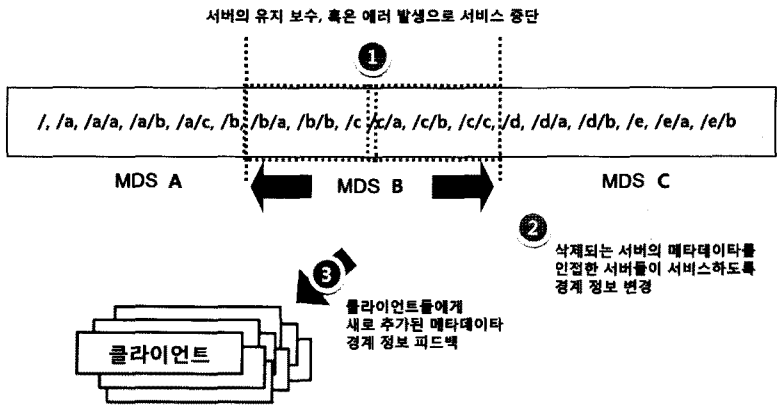


그림 10 메타데이타 서버의 삭제

표 2 실험 환경

실험환경	값
CPU	Intel Pentium4 3.0GHz Zeon CPU
RAM	2GB
HDD	SAS 74GB
OS	Linux Fedora core6(Kernel v.2.6.10)
Ethernet	10GB

표 3 성능 평가 파라미터

파라미터	값
서버 수	8대
시스템 디렉토리 계층 구조	Level 5 (/L5D1/L4D1/L3D1/L2D1/L1D1)
디렉토리 별 기본 파일 수	100개
총 메타데이타 수	1175706개 (디렉토리 : 3906, 파일 : 1171800)
서버 당 동시 접근 허용 횟수	100회

이터베이스 제어 및 모든 연산은 C 언어로 구현되며, 서버들 간의 모니터링 및 메타데이타 관리를 위한 통신은 TCP로 수행된다.

각 메타데이타 서버는 표 3의 성능평가 파라미터에 따라 기본적인 데이터를 보유하고 있다. 실험에 사용한 데이터는 High Performance Computing(Lawrence Livermore National Laboratory)에서 제공하는 Scalable I/O Benchmark를 위한 메타데이타 생성 툴인 FDTree(Ver 1.0.2)를 통해 작성되었다[14].

먼저 각 서버들은 초기에 균등한 수의 메타데이타를 갖도록 설정되며, 서버 수에 따라 유지하는 메타데이타의 수와 디렉토리 분할 위치는 적절하게 조정한다. 메타데이타 서버는 클라이언트 당 하나의 쓰레드(thread)로 서비스하며, 최대 100회의 동시 접근을 처리할 수 있다. 각 메타데이타 서버에서 관리하는 메타데이타 정보는 Unix 계열 시스템의 메타데이타 구조에 비대칭형 클러스터 구조를 갖는 파일시스템의 메타데이타 특징을 고려하여 설계하였다.

제안하는 기법(DicPS)의 우수성을 보이기 위해 대표적인 비대칭형 클러스터 파일시스템을 위한 비공유 메타데이타 관리 기법인 해시(HASH), Lazy Hybrid(LH), 동적 서브트리 분할(DSP)과 비교 평가를 수행하였다. 성능 평가는 메타데이타 요청 횟수 및 메타데이타 서버 수 대한 타 기법과의 비교 평가와 일시적인 부하 및 지속적인 부하 시, 제안하는 기법의 성능 변화에 대해 평가를 수행하였다.

4.2 메타데이타 요청 횟수에 따른 성능 평가

메타데이타 서버 클러스터에 다수의 클라이언트들이 요청한 연산의 수에 따른 메타데이타 서버의 평균적인 응답 시간에 대한 실험을 수행하였다. 실험은 4대의 메타데이타 서버로 구성된 클러스터에서 읽기(read) 연산, 쓰기(write) 연산, 읽기-쓰기 연산으로 나누어 수행하였다. 읽기-쓰기 연산의 경우 그 비율을 일반적인 2:8 비율에 따라 80%의 읽기 연산과 20%의 쓰기 연산으로 구성하였다. 각 연산은 1,000개의 클라이언트 기능을 수행하는 쓰레드에 의해 분할 수행되며, 각 연산은 미리 구축한 파일시스템 네임스페이스에서 임의의 디렉토리를 선택하여 수행된다.

제안하는 기법(DicPS)은 각 메타데이타 서버가 관리하는 메타데이타 정보가 그 계층 구조를 유지하도록 분할 관리한다. 따라서 메타데이타 연산 시, 접근한 메타데이타에 대한 참조 지역성을 최대한 이용할 수 있다. 또한 클라이언트가 임의의 메타데이타 서버에 접근할 경우, 현재 메타데이타 서버 클러스터 내의 메타데이타 배치 정보를 분할 기준 정보로 제공하기 때문에, 한 번 메타데이타 서버 클러스터에 접근한 클라이언트들은 이후 작업 수행 시, 메타데이타 서버 간의 추가적인 메시지 전달 연산 없이 바로 요청한 메타데이타를 관리하는 메타데이타 서버에 바로 접근할 수 있다.

그림 11, 12, 13은 각각 메타데이타 수에 따른 성능 비교 결과를 보인다. 실험 결과, 모든 연산에서 제안하는 기법이 타 기법에 비해 가장 좋은 성능을 보인다. Hash와 LH와 같이 해시 계열의 메타데이타 기법은 메타데이타 정보의 계층 구조를 유지할 수 없기 때문에 추가적인 메타데이타 서버 순회 비용이 요구된다.

동적 서브트리 기법과 제안하는 기법은 한 서버에서 관리되는 메타데이타 정보가 계층 구조를 최대한 유지하며 관리되기 때문에 추가적인 서버 순회 비용을 크게 감소시킬 수 있다. 또한 제안하는 기법은 동적 서브트리의 단점인 메타데이타 서버 간의 추가적인 메시지 전달 비용을 메타데이타 분할 기준 정보를 통해 최소화하여 타 기법에 비해 최소 10~30% 이상의 성능 향상을 보였다.

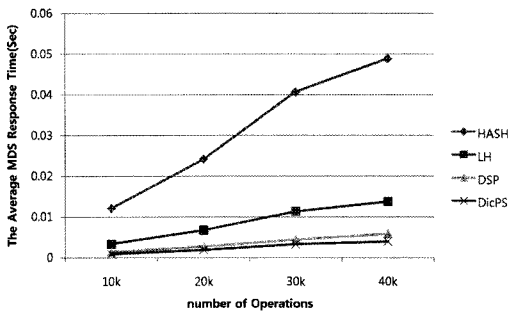


그림 11 메타데이타 요청 횟수에 따른 응답 시간 (읽기 연산)

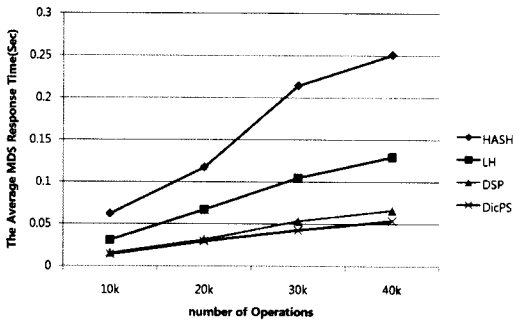


그림 12 메타데이타 요청 횟수에 따른 응답 시간 (쓰기 연산)

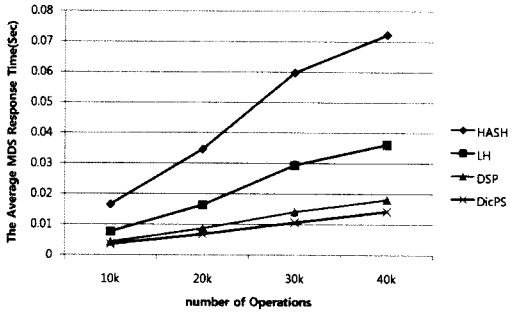


그림 13 메타데이타 요청 횟수에 따른 응답 시간 (읽기-쓰기 연산)

4.3 메타데이타 서버 확장에 따른 성능 평가

대용량 파일시스템에서 시스템을 구성하는 메타데이타 서버의 확장은 필수적이다. 시스템이 관리하는 메타데이타 수의 증가 및 이를 이용하는 사용자들에게 원활한 서비스를 제공하기 위해서 적절한 수의 메타데이타 서버가 추가될 수 있다. 이에 메타데이타 서버 수 증가에 따른 메타데이타 서버 클러스터의 평균 응답 시간을 각각 읽기(read) 연산, 쓰기(write) 연산, 읽기-쓰기 연산으로 나누어 수행하였다. 읽기-쓰기 연산의 경우 그 비율을 일반적인 2:8 비율에 따라 80%의 읽기 연산과 20%의 쓰기 연산으로 구성하였다. 각 연산은 1,000개의 클라이언트 기능을 수행하는 쓰레드에 의해 분할 수행되며, 총 40,000개의 연산을 수행한다. 각 연산은 미리 구축한 파일시스템 네임스페이스에서 임의의 디렉토리를 선택하여 수행된다.

그림 14, 15, 16은 각각 메타데이타 서버 수에 따른 성능 비교 결과를 보인다. 실험 결과, 모든 연산에서 제안하는 기법이 타 기법에 비해 가장 좋은 성능을 보인다. 서버가 추가되면서 메타데이타가 여러 서버로 분할 관리되고, 클라이언트의 요청을 다수의 서버가 처리하면서, 평균적인 응답시간이 모든 기법에서 개선됨을 확인

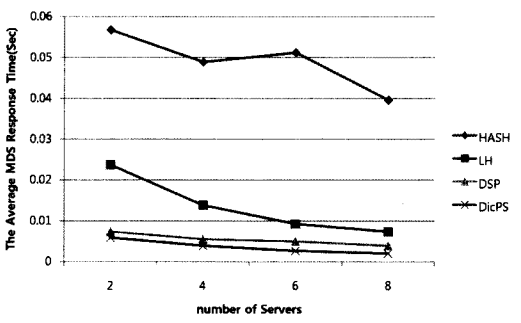


그림 14. 메타데이타 서버 수에 따른 응답 시간 (읽기 연산)

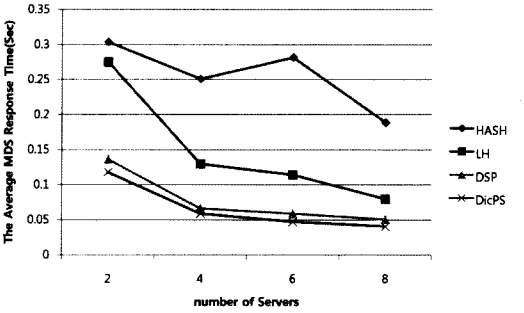


그림 15 메타데이타 요청 횟수에 따른 응답 시간 (쓰기 연산)

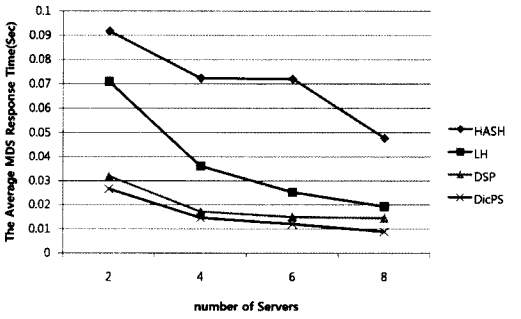


그림 16 메타데이타 요청 횟수에 따른 응답 시간 (읽기-쓰기 연산)

하였다. Hash의 경우, 연산에 사용된 메타데이타가 배포된 메타데이타 서버 수에 따라, 일부의 경우 서버 추가에 따른 응답 시간이 악화되는 경우가 발생하였다. LH의 경우 메타데이타 서버 수 추가에 따라 응답 시간이 상당히 개선됨을 확인할 수 있다. 이는 연산에 사용된 메타데이타 정보의 구성에 따라, ACL이 적용되면서 서브트리 기법과 같이 참조 지역성을 활용할 수 있기 때문으로 풀이된다. 그러나 LH의 경우, 디렉토리의 권한 변경 등과 같은 연산이 같이 고려되면, 이미 배포된

ACL 정보의 수정이 불가피하기 때문에 현재의 성능이 크게 악화될 수 있다. 동적 서브트리 기법과 제안하는 기법은 서버 추가에 따라 그 응답시간이 향상되었다. 그러나 두 기법 모두 서브트리를 기반으로 하는 메타데이타 관리 기법이기 때문에, 서버 추가에 따른 큰 성능 차이는 보이지 않는다. 그러나 메타데이타 서버 클러스터의 규모가 커지면서, 클라이언트가 임의의 메타데이타 서버에 접근했을 때, 요청한 메타데이타의 부재로 메타데이타 서버 간의 추가적인 메시지 전달이 발생할 가능성은 더욱 높아진다. 따라서 보다 정확한 메타데이타 분할 정보를 제공하는 제안하는 기법이 보다 좋은 성능을 보인다.

4.4 복제 정보를 사용한 일시적인 부하 분산 성능 평가

특정 메타데이타 서버에 저장된 단일 메타데이타 정보에 대해 빈번한 접근 요청이 발생하는 경우 이 메타데이타 정보의 복제 정보를 만들어 접근 요청의 응답시간을 줄일 수 있다. 즉 하나의 메타데이타 서버에 부하가 집중되는 것을 방지하기 위하여 동일한 메타데이타 정보를 여러 메타데이타 서버에서 유지함으로써 읽기 작업에 관한 부하를 경감시킬 수 있다. 따라서 제안하는 메타데이타 관리기법에서 복제 정보를 사용한 경우의 평균 작업 처리량을 평가하였다.

실험 환경은 8대의 메타데이타 서버로 구성된 시스템에서 40,000건의 읽기 연산이 연속적으로 수행되는 상황을 고려하였다. 수행되는 읽기 연산 중 80% 이상을 특정 서버의 단일 메타데이타에 대한 읽기 요청으로 수행하였다. 실험 결과는 복제 정보를 생성했을 때 메타데이타 서버 클러스터의 평균 작업 처리량과 복제 정보를 생성했을 때 각 메타데이타 서버 간 작업 처리량을 비교한다.

그림 17과 18은 일시적인 읽기 부하에 따른 전체 클러스터의 작업 처리량과 메타데이타 서버 간 작업 부하 범위를 보인다. 실험 결과, 약 5초 이후 평균 작업 처리량이 개선되었음을 확인할 수 있다. 즉 실험을 수행한 지 5초 시점에서 메타데이타에 대한 일시적인 부하를 감지한 서버가 메타데이타 정보에 대한 복제를 타 서버에 요청했고, 이후 전체 메타데이타 서버 클러스터의 평균 작업 처리량이 향상됐음을 실험 결과를 통해 확인할 수 있다. 또한 복제가 이루어지는 5초 전까지 클러스터 내의 메타데이타 서버 간 작업 처리량은 매우 심한 편차를 보였으나, 약 9초 전후에서 모든 서버에 메타데이타에 대한 복제가 수행된 후에는, 매타데이터 복제를 통해 그 격차를 상당부분 완화됐음을 확인할 수 있다.

비공유 메타데이타 관리 기법은 각 메타데이타 서버가 서버 내에 저장된 메타데이타만 관리하기 때문에, 특정 서버에 메타데이타 요청이 집중되면 그림 12의 초기

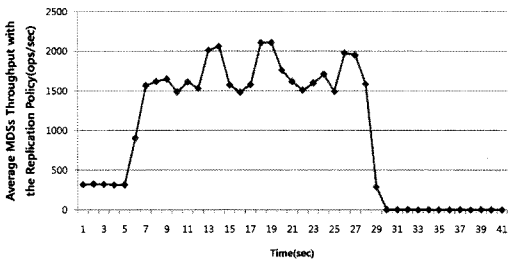


그림 17 일시적인 부하(읽기)에 따른 메타데이타 서버 클러스터 작업 처리량

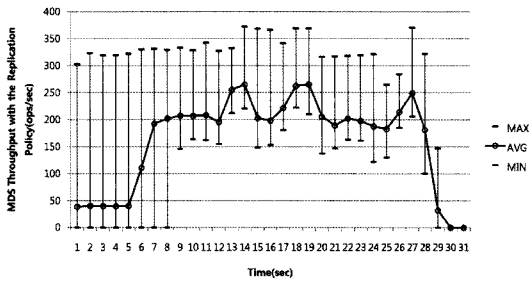


그림 18 일시적인 부하(읽기)에 따른 메타데이타 서버 간 작업 부하 범위

처리량과 같이 서버 간 작업 부하 범위가 매우 큰 차이를 보인다. 제안하는 기법은 다른 메타데이타 서버에 일시적인 부하가 집중되는 메타데이타에 대한 복제 정보를 배포하여 해당 메타데이타에 대한 요청을 대신 처리한다. 따라서 일시적인 부하에서도 특정 메타데이타 서버로 작업 요청이 집중되는 것을 효과적을 방지할 수 있다.

4.5 분할 정보 재조정을 사용한 지속적인 부하 분산 성능 평가

제안하는 메타데이타 관리기법에서 지속적인 쓰기 부하에 대한 부하 분산 정도를 평가하기 위해 단일 서버에 쓰기 부하을 집중시켜 메타데이타 서버 간 메타데이타 수의 불균형을 강제로 발생시켰다. 그리고 메타데이타 서버 클러스터의 평균 작업 처리량과 각 메타데이타 서버 간 작업 처리량을 비교하였다.

실험 환경은 8대의 메타데이타 서버로 구성된 시스템에서 40,000건의 쓰기 연산이 연속적으로 수행되는 상황을 고려하였다. 수행되는 쓰기 연산 중 80% 이상을 특정 서버의 최하위 디렉토리에서의 파일 생성 요청으로 수행하였다. 제안하는 메타데이타 관리기법은 특정 서버의 메타데이타 수가 전체 시스템의 평균 메타데이타 수의 10% 이상 증가하게 되면, 분할 기준을 재조정한다고 가정하였다. 실험 결과는 분할 기준을 재조정하였을 때 메타데이타 서버 클러스터의 평균 작업 처리량과 각 메타데이타 서버 간 작업 처리량을 비교한다.

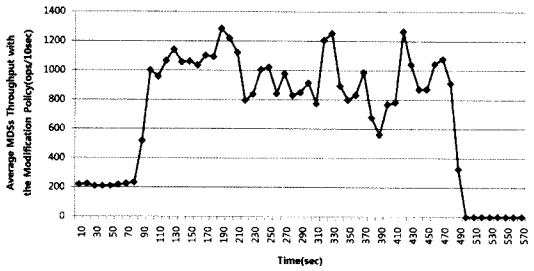


그림 19 지속적인 부하(쓰기)에 따른 메타데이타 서버 클러스터의 작업 처리량

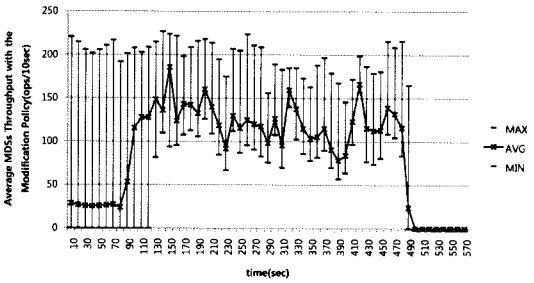


그림 20 지속적인 부하(쓰기)에 따른 메타데이타 서버 간 작업 부하 범위

그림 19와 20은 지속적인 쓰기 부하에 따른 전체 클러스터의 작업 처리량과 메타데이타 서버 간 작업 부하 범위를 보인다. 실험 결과 80초 이후 특정 서버에 집중됐던 메타데이타 생성 요청이 타 메타데이타 서버로 분산되는 것을 확인할 수 있다. 이는 특정 디렉토리에서의 빈번한 메타데이타 생성을 감지한 후, 해당 디렉토리에 대한 메타데이타 분할 요약 정보를 일부 분할하여, 타 서버에 위임했기 때문이다. 따라서 110초 이후에는 타 서버로 메타데이타 분할 경계의 조정을 모두 통보하여, 전체 메타데이타 클러스터의 작업 처리량을 향상시킨다.

5. 결 론

최근 대용량 스토리지 및 관리 기술에 대한 요구가 증가하면서 대용량 클러스터기반 분산파일시스템 및 이와 관련된 기술들에 대한 다양한 연구가 활발하게 진행되고 있다. 본 논문에서는 최근 많은 연구가 진행되고 있는 대용량 클러스터 스토리지 시스템 분야에서 대용량 메타데이타를 효과적으로 관리하고, 시스템 확장에 따른 충분한 확장성과 성능 향상을 제공하기 위한 메타데이타 관리 기법을 제시하였다.

본 논문에서는 기존 메타데이타 관리 기법의 비교 분석을 통해 사전식 분할 기법과 분할 기법을 지원하는 부하 분산 기법을 설계하였으며, 설계한 메타데이타 관리 기법을 적용한 메타데이타 서버 클러스터를 구성하여 타 비공유 메타데이타 관리 기법을 적용한 서버 클러스터와 성능 비교 평가를 수행하였다. 성능 평가 결과 제안하는 기법을 적용한 비공유 메타데이타 서버기 타 기법을 적용한 메타데이타 서버 대비 대비 10~30% 연산 성능이 향상되었다. 또한 복제 정보 및 분할 정보의 동적 조정을 사용하여 시스템의 부하를 효과적으로 처리하고 있음을 보였다.

향후 제안하는 시스템의 결함 허용 정책을 보완하고, 메타데이타 정보 및 색인 정보를 보다 효과적으로 저장할 수 있도록 추가적인 연구를 진행할 것이다. 또한 시스템의 부하 판별 기준을 보다 명확히 하여 다양한 작업 부하에 효과적으로 대처할 수 있는 메타데이타 서버 클러스터를 연구할 것이다.

참 고 문 헌

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

[2] K. W. Preslan et al., "A 64bit, Shared Disk File System for Linux," *Proceedings of the 16th IEEE Mass Storage Systems Symposium*, pp.22-41, 1999.

[3] M. Mesnier, G. Ganger, and E. Riedel, "Object based Storage," In *IEEE Communications Magazine*, pp.84-90, 2003.

[4] Sage Weil, Scott A. Brandt, Ethan L. Miller, Darrell D.E. Long, and Carlos Maltzahn, "Ceph : A Scalable, High-Performance Distributed File System," In *Proc. of Conf. on Operating Systems Design and Implementation*, pp.307-320, 2006.

[5] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System," In *Proc. of ACM Symp. on Operating Systems Principles*, pp.20-43, 2003.

[6] D. Borthakur, "The Hadoop Distributed File System: Architecture and Design," 2005

[7] Lustre, "Lustre : A Scalable High Performance File System," Cluster File System Inc., 2002.

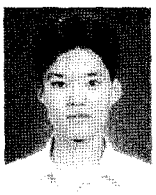
[8] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas Active Scale Storage Cluster-Delivering Scalable High Bandwidth Storage," In *Proc. of the ACM/IEEE SC2004 Conf.*, 2004.

[9] 민영수, 차명훈, 김영철, 진기성, 이상민, 정병권, 김준, "객체기반 저장 장치를 이용한 클러스터 파일시스템의 구현", *한국차세대컴퓨팅학회논문지*, 제2권, 제4호, pp.42-52, 2006.

[10] 차명훈, 이상민, 김준, 김영균, 김명준, "대규모 분산 파일 시스템 환경의 메타 데이타 관리", *전자통신동향분석*, 제22권, 제3호, pp.154-165, 2007.

[11] S.A. Brandt, L. Xue, E.L. Miller, and D.D.E. Long, "Efficient Metadata Management in Large Distributed File Systems," In *Proc. of IEEE/11th NASA Goddard Conf. on Mass Storage Systems and*

Technologies, 2003.
[12] Sage Weil, Kristal Pollack, Scott A. Brandt, and Ethan L. Miller, "Dynamic Metadata Management for Petabyte-Scale File Systems," In *Proc. of the ACM/IEEE Conf. on Supercomputing*, 2004.
[13] 장준호, 한세영, 박성용, "클러스터 파일시스템의 메타데이타 서버를 위한 내용 기반 부하 분산 알고리즘", *정보처리학회논문지*, 제13권, 제4호, pp.323-334, 2006.
[14] FDTree, "https://computing.llnl.gov/?set=code&page=sio_downloads"



윤 종 현

2003년 충북대학교 전기전자및컴퓨터공학부 정보통신공학(공학사). 2005년 충북대학교 정보통신공학과(공학석사). 2009년 충북대학교 정보통신공학과(공학박사) 2009년 2월~현재 한국전자통신연구원 관심분야는 DBMS, 저장 시스템, 시공간 색인 구조, 이동 객체 DBMS, 센서 네트워크 등

박 용 훈

2005년 호원대학교 정보통신공학과 및 건축공학과(공학사). 2007년 충북대학교 정보통신공학과(공학석사). 2007년~현재 충북대학교 정보통신공학과 박사과정. 관심분야는 데이타베이스 시스템, 정보검색, 시공간 데이타베이스, 센서 네트워크 등

이 석 재

2000년 충북대학교 정보통신공학과(공학사). 2002년 충북대학교 정보통신공학과(공학석사). 2006년 충북대학교 정보통신공학과(공학박사). 2006년~2007년 7월 충북대학교 BK21 Post-Doc. 2007년 8월~2008년 5월 애플테크(주) 기술연구소장. 2008년 6월~현재 한국전자통신연구원. 관심분야는 DBMS, 저장 시스템, 주기억장치 DBMS, 센서 네트워크 등

장 수 민

1997년 목포대학교 전산통계학과(학사) 1999년 충북대학교 정보통신공학과(석사) 2007년 충북대학교 정보통신공학과(박사) 2007년 충북대학교 초빙전임강사 2007년~현재 충북대학교 BK21 Post. Doc 관심분야는 게임서버, 정보검색, 분산 객체 컴퓨터 등

유 재 수

1989년 전북대학교 컴퓨터공학과(학사) 1991년 한국과학기술원 전산학과(석사) 1995년 한국과학기술원 전산학과(박사) 1996년 충북대학교 전기전자컴퓨터공학부 부교수. 2006년~현재 충북대학교 전기전자컴퓨터공학부 교수. 관심분야는 데이타베이스시스템, 멀티미디어 데이타베이스시스템, 정보검색, 분산 객체 컴퓨터 등

김 홍 연

1992년 인하대학교 통계학과(이학사). 1994년 인하대학교 전자계산학과(공학석사) 1999년 인하대학교 전자계산학과(공학박사). 1999년 8월~현재 한국전자통신연구원. 관심분야는 네트워크스토리지및파일시스템, 클러스터컴퓨팅, DBMS

김 영 균

1991년 전남대학교 전산통계학과(이학사) 1993년 전남대학교 대학원 전산 통계학과(이학석사). 1995년 전남대학교 대학원 전산 통계학과(이학박사). 1995년~현재 한국전자통신연구원. 관심분야는 클러스터 DBMS, 클러스터 파일 시스템, 멀티미디어 파일 시스템