Journal of Digital Contents Society Vol. 16 No. 2 Apr. 2015(pp. 299-306)
http://dx.doi.org/10.9728/dcs.2015.16.2.299

분산 파일 시스템 기반 NoSQL의 데이터 안정성을 위한 범용 트랜잭션 관리 기법

권영현*, 윤도현**, 박호진***

요 약

본 논문에서는, 분산 파일 시스템을 기반으로 하는 NoSQL의 데이터 안정성 확보를 연구하였다. 본 논문의 궁극적 목표인 분산 파일 시스템 기반의 NoSQL을 구현하는 과정에서 분산 파일 시스템의 제약 조건인 랜덤 쓰기 문제에 봉착했고, 이 문제의 해결을 위해 중간파일의 개념을 사용함으로써 어떠한 장애 상황에서도 데이터의 오염을 방지할 수 있었다. 또한 중간파일을 쓰는 과정에서 기존 파일 시스템에 비해 분산 파일 시스템이 가지는 성능적 열세를 인식하여, NoSQL을 위한 파일 블록 단위를 다시 정의하는 방법으로 성능적 손실을 크게 줄였다. 결과적으로, 본 논문에서는 보편적 분산 파일 시스템의 확장성을 가진 NoSQL을 개발함과 동시에 원자성, 일관성, 고립성, 성능 등의 조건을 만족하는 트랜잭션 관리 기법을 사용함으로써 데이터 안정성을 가지면서 실용적 사용에도 무리가 없는 NoSQL을 구현하였다.

키워드 : 데이터 안정성, 분산 파일 시스템, 빅데이터, 트랜잭션, NoSQL.

General-purpose Transaction Management Technique for Data Stability of NoSQL on Distributed File System

Younghyun Kwon*, Do-hyun Yun**, Hojin Park***

Abstract

In this paper, we research to secure stability of data storing/searching on NoSQL implemented on Distributed File System. When implementing NoSQL on Distributed File System, we faced that random write on Distributed File System is almost impossible. To solve this problem, a concept of Intermediate-File was employed, and then it has been achieved that our system resist any failure circumstance. Additionally, since we discovered its performance cannot be as fast as general File System, by redefining the file block unit for our NoSQL system, we have prevented a slowdown in system performance. As a result, we are able to develop highly scalable NoSQL as Distributed File System, which fulfills basic conditions of transaction: Atomicity, Consistency, Isolation, and Performance.

Keywords : Big data, Data Stability, Distributed File System, NoSQL, Transaction.

1. 서론

우리는 날마다 데이터의 홍수 속에 살고 있다. 데이터는 1980년대 이후로 매 40주마다 2배씩 증가해왔으며[1], 2012년 기준으로 매일 2.5 엑사바이트의 데이터가 생산되고 있다[2]. 이와 같이 20세기 말부터 시작된 데이터의 폭증은 전통적 데이터 처리 방법론에 제동을 걸었다. 데이터는 넘쳐났으나 개별 컴퓨터에 종속된 데이터 저장

※ Corresponding Author : Hojin Park
Received : March 13, 2015
Revised : April 21, 2015
Accepted : April 30, 2015
* Emerging Technology R&D Center, WISEnut, Inc.
** Emerging Technology R&D Center, WISEnut, Inc.
*** Emerging Technology R&D Center, WISEnut, Inc.
Tel: +82-2-3404-6134 , Fax: +82-2-3404-6109
email: hanwool@wisenut.co.kr
▣ 본 연구는 미래창조과학부 및 정보통신기술연구진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음. [10045341 , 글로벌 경쟁력을 갖춘 기업용 페타스케일급 정형 및 비정형 빅데이터 검색/분석 SW 개발]

소와 물리적 저장 공간의 지속적인 확장이 불가능하였고, 데이터 증가에 유연한 대처가 가능한 수평적 확장 구조를 가지는 데이터 저장소의 필요성이 대두되었다.

GFS[3](Google File System)는 비교적 저렴한 하드웨어를 사용하여 수평적 확장에 용이한 분산 파일 시스템을 제안하였고, HDFS[4](Hadoop File System)를 비롯한 후발 주자들이 이러한 움직임에 동참하였다. 그리고 파일 저장 수준의 확장성과 물리적 저장소의 확장성을 고려한 분산 파일 시스템과는 별개로, 종래의 관계형 데이터베이스에 대응하는 데이터 관점의 수평적 확장에 주안점을 두는 NoSQL 또한 주목받기 시작했다.[5] 현대의 NoSQL은 전통적인 관계형 데이터베이스의 SQL문법과 유사한 사용성을 가지며, 그 기능 일부를 제약하는 대신 대용량 데이터를 다루는데 근본적 의의[6] 를 지니며 성장과 관심을 받아오고 있다.

이 과정에서 분산 파일 시스템이 가진 물리적인 확장 용이성을 활용하여 Apache HBase, Oracle NoSQL, Cloudera Impala와 같이 분산 파일 시스템과 통합된 형태의 NoSQL도 등장하기에 이르렀다.

본 논문에서는, 이러한 시류에 맞추어, 거시적 관점에서 수평적 확장성이라는 공통분모를 가지는 두 시스템의 통합 과정을 설명하고 필연적으로 발생할 수밖에 없는 데이터 안정성 저하의 문제점을 살펴보고 해결책을 연구한다.

2. 관련연구

2.1 분산 파일 시스템

분산 파일 시스템은 네트워크 기반의 파일 시스템으로서, Rani et al.[7] 은 시스템 설계 고려사항으로 사용자가 마치 일반적인 파일 시스템을 사용하고 있는 것과 마찬가지의 인식만을 가지게 해야 한다는 관점에서의 투명성과 커널에 종속적이지 않아야 한다는 유연성을 설명했다. 또 어떠한 에러 상황에서도 접근이 가능해야 한다는 신뢰성, 그리고 단일 프로세서에서 사용하는 것에 비할만한 성능을 갖춰야 한다는 조건을 기술했으며, 허용되지 않은 접근이나 그를 통한 데이터 오염에 대한 보안 문제의 해결에 대한 필요도 언급했다.

그리고 근래에 분산 파일 시스템에 대한 관심을 불러일으킨 주역인 확장 용이성은 종래의 중앙 집중형 시스템의 한계를 드러냄으로써 유연한 확장에 대한 필요성을 간접적으로 피력했다. 마지막으로 분산된 컴퓨터 환경에서 일부 컴퓨터에 대한 실패에도 정상동작하도록 해야 한다는 실패 관용성을 들었다.

<표 1> 투명성의 종류

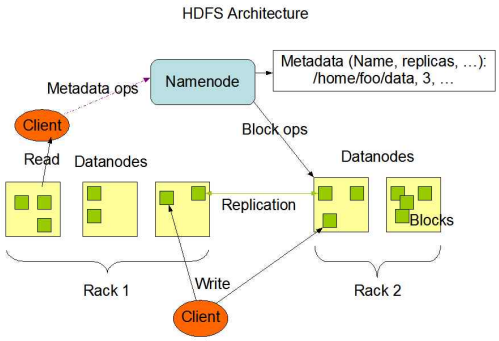
Transparency	Description
Access	Hides the differences in data representation and how a resource is accessed.
Location	Hides the physical location of a resource.
Migration	Hides the movement of a resource to another location.
Relocation	Hides the movement of a resource to another location while in use.
Replication	Hides the fact that multiple copies of the resource exist without user's knowledge

<Table 1> Types of Transparencies [8]

데이터의 절대적인 크기가 증가함에 따라, 대용량의 데이터를 저장하는 동시에 그것을 처리할 수 있는 강력한 연산 능력의 필요성이 대두되었다. 이에 부응하여 Google은 Map-Reduce[9]를 통한 다중 연산 능력을 활용할 수 있는 GFS를 선보였으며, 이후 등장한 Apache Hadoop의 HDFS는 Yahoo의 전폭적인 지지와 오픈 소스 진영의 노력으로 인해 대중에게 보편적인 대용량 분산 파일 시스템으로 인식되었다.

HDFS의 실제 데이터를 담는 Datanode는 논리적 파일의 파편인 고정 길이의 블록을 물리적인 파일들로 저장한다. 물리적 파일들은 각자의 복사본(Replica)을 보관하여 실패 관용성을 가지며 이 복사본들의 위치는 Namenode에 의해 논리 파일의 메타 정보와 함께 관리되고 클라이언트의 요청에 의해 참조된다. (그림 1)은 이 과정을 도식화 한 것이다.

(그림 1) Apache HDFS 아키텍처



(Figure 1) Apache HDFS Architecture[10]

HDFS는 Datanode의 물리적 추가와 Namenode로의 등록을 통해 데이터 저장 용량을 손쉽게 확장할 수 있도록 하였고, 복사본을 활용하여 시스템의 신뢰성을 보장하는 것과 더불어 실패 상황에 대한 대비를 하였다. 또 기존의 파일 시스템과 대부분의 인터페이스 호환을 통해 투명성을 유지하고 클라이언트의 시스템에 종속되지 않는 네트워크 I/O를 통하여 유연성을 충족시켰다.

2.2 NoSQL

Carlo[11]에 의한 초기의 NoSQL이라는 용어는, 단순히 그 자신이 만든 표준 SQL 인터페이스를 사용하지 않는 관계형 데이터베이스를 정의하기 위해 사용했다. 그러나 근래에 들어서 급부상하는 대용량 데이터 처리의 요구에 따라 종래의 관계형 데이터베이스의 한계인 수평적 확장 용이성을 극복하며, 탈관계형 데이터 관리 시스템을 뜻하는 단어로 변화하고 있다고 밝혔다.

빅데이터 처리에 대한 요구와 기대감으로 21세기 초부터 다양한 제품 NoSQL이 다양하게 발표되었는데, Gajendran[12] 는 그의 NoSQL 조사 논문에서 다양한(키-값, 문서 혹은 컬럼)기반의 NoSQL인 BigTable, Cassandra, HBase, MongoDB 등의 제품들을 소개하면서 각 제품의 특성을 설명하고 분류함과 더불어, NoSQL에 등장하는 스키마 수평 분할(Sharding), 일관된 해싱 기법, 다중 버전 동시제어(MVCC) 혹은 Locking 기법 등의 공통적 특성에 대해서도 설명했다.

Stonebraker[13] 는 성능적 측면에서 NoSQL과 SQL을 비교하면서, 온라인 프로세스 트랜잭션 처리(OLTP:Online Transaction Processing)의 속도를 예로 두 시스템에 추구하는 트랜잭션 개념의 차이를 기술했으며, 간접적으로 NoSQL의 트랜잭션에서 ACID(원자성, 일관성, 고립성, 지속성) 조건을 충족시키기 불리하기 때문에 RDBMS의 대체가 어려울 것이라 전망했다.

이러한 전망에도 불구하고 ACID 조건을 충족하는 NoSQL을 만들고자 하는 노력도 지속되어 왔다. Wei[14] 는 클라우드 기반의 NoSQL의 분산 트랜잭션 기법에 대해 연구했고, Google의 Peng[15] 은 웹문서에 대한 증분 색인을 위한 동적 수집, 색인(변경된 웹페이지 수집, 색인)을 위한 분산 트랜잭션 기법을 소개했다.

비록 기존에 서비스되고 있는 OLTP 응답성에는 미치지 못할 수 있지만 ACID 조건을 충족하는 NoSQL은 지속적으로 개발되고 있으며, 이에 대한 요구는 점차 강해지고 있다고 전망할 수 있다.

3. 분산 파일 시스템 기반의 NoSQL 문제점

수평적 확장성은 분산 파일 시스템과 NoSQL이 가지는 공통적 목적이며 기본적으로 보유하고 있는 속성이다. 바꾸어 말해, 분산 파일 시스템 기반으로 NoSQL을 구축한다면 적어도 확장에 용이한 성질은 그대로 상속될 것이라는 것은 쉽게 예상할 수 있다.

본 논문은 분산 파일 시스템의 확장 용이성을 활용한 문서기반의 NoSQL 시스템을 구축하는 과정에서 분산 파일 시스템의 제약조건으로 인한 데이터 안정성 측면의 문제를 인식하고 원자성, 일관성, 고립성의 조건을 만족하는 트랜잭션 기법에 대해 연구했다. 본 장에서는 분산 파일 시스템에 NoSQL을 접목할 때 발생할 수 있는 문제를 설명하고 다음 장에서 본 논문의 접근 방식으로 해결 방안을 제시한다.

3.1 분산 파일 시스템의 랜덤 쓰기 제약

로컬 파일 시스템에서의 쓰기 작업과 다르게, 대용량 데이터 저장과 잦은 읽기에 최적화되어 있는 분산 파일 시스템에서는 랜덤 쓰기 작업을

허용할 경우, 시스템에 미치는 영향은 매우 크다. 이것의 근본적인 원인은 (1)하나의 논리적 파일이 여러 개의 물리적 파일로 쪼개져 네트워크 기반으로 분산되어 있고, (2)데이터 가용성과 신뢰성과 실패 관용성을 위해 복사본(Replica)을 만들기 때문이다.

로컬 파일 시스템의 파일에서 오프셋을 건너뛰어 쓰기 작업을 하는 행위를 디스크 헤드의 움직이는 거리를 늘린다는 것으로 단순화 할 수 있다면, 분산 파일 시스템의 그것은 (1)오프셋과 파일 블록간의 매핑을 수시로 조회하는 작업과 (2)복사본간 동기화 작업이 추가되어야 할 것이라 쉽게 예상할 수 있다. 특히 랜덤 쓰기에 대한 복사본 동기화 작업은 쓰기 성능을 떨어뜨릴 뿐만 아니라 파일을 읽는 작업의 안정성에도 치명적이다.

랜덤 쓰기가 가능한 분산 파일 시스템을 가정해보자. 하나의 논리적 파일이 n개의 물리적인 파일(블록)로 나뉘어 있고 이 물리적인 파일은 각각 m개의 복사본을 갖는다. 여기에 수시로 랜덤 읽기 작업(r)을 하는 p개의 프로세스 $r_1,...,r_p$가 있다. 랜덤 쓰기 작업을 하는 또 다른 프로세스 w는 특정 바이트 단위로 건너뛰며 1바이트씩 데이터를 수정한다. r_a과 r_b가 동시에 k번째 파일 블록(b)의 각각 다른 복사본 b_{ki}, b_{kj}에 접근하려는 순간 프로세스 w가 쓰기 작업을 수행한다고 생각해보자. 여기서 w가 쓴 데이터가 b_{ki}, b_{kj}에 동시에 적용됨을 보장할 수 없다는 사실에 주목하자. 두 블록은 물리적으로 분리된 파일이므로 필연적으로 r_a, r_b은 다른 데이터를 읽는 상황이 발생할 수 있으며, 이는 시스템의 데이터 일관성을 깨뜨린다. 혹자는 로컬 파일 시스템에서도 여러 프로세스가 읽기 작업을 하는 동안 데이터 쓰기가 일어난다면 r_a, r_b가 읽은 데이터가 다를 수 있는 것은 마찬가지가 아니냐고 말할 수 있다. 하지만 그것은 각 프로세스의 읽기 작업 수행의 시간 차 때문이지, 데이터의 쓰기가 물리적 파일에 적용되는 시점에 의한 것이 아니다. 이것은 데이터 적용이 이루어 질 때 로컬 파일 시스템은 유일한 파일을 쓰지만, 복사본을 관리하는 분산 파일 시스템은 여러 파일을 쓰기 때문에 읽는 시점과 위치에 따라 데이터 간 불일치가 발생하는데, 이는 본 논문이 추구하는 NoSQL의 기반 파일시스템으로써 갖추어야할 고립성(Isolation) 성질을 충족시킬 수 없는 제약사항으로 볼 수 있다.

반면에 랜덤 쓰기를 금지하는 분산 파일 시스템은 어떻게 다를까? 비록 각 복사본에 데이터를 쓰는 시점이 다를 수 있다는 것은 분명하다. 그러나 항상 파일의 가장 마지막 부분에만 데이터 쓰기가 일어나므로 모든 복사본의 데이터가 각 물리적 복사본에 완전히 저장될 때까지 해당 오프셋에 대한 접근을 금지하는 것만으로 데이터 일관성을 유지할 수 있음을 알 수 있다. 이처럼 데이터의 안정성을 저해하는 이유로, HDFS 및 GFS는 랜덤 쓰기를 원천적으로 지원하지 않고 있다.

3.2 NoSQL의 데이터 신뢰성

NoSQL의 본 목적은 데이터 저장/검색, 그 자체이므로 데이터의 안전한 저장은 NoSQL의 그 어떤 기능적 요소보다 중요하다고 할 수 있다. 그 중에서도 특히, 데이터를 쓰는 시점에서 발생할 수 있는 모든 장애 상황에 대하여 데이터 무결성을 보장하는 것은 비단 NoSQL뿐만 아니라 전통적인 관계형 데이터베이스로부터 이어져 내려오는 데이터 저장소의 최우선 과제임은 의심의 여지가 없다.

분산 파일 시스템의 랜덤 쓰기 제약은 NoSQL의 데이터 신뢰성 측면에서 심각한 문제점을 발생시킬 수 있다. 랜덤 쓰기가 불가능하다는 제약조건은 이미 물리적으로 한번 저장된 파일은 지우고 다시 쓰지 않는 한, 물리적 파일의 내용을 부분적으로 변경, 삭제할 수 없음을 뜻한다. 좀 더 직접적으로 설명하자면 데이터를 쓰는 작업 도중 쓰는 작업을 하는 프로세스가 어떤 이유에서건 작동이 중지되면 그 파일은 써야할 내용의 일부만 저장되어 있는 상태가 된다. 그렇게 되면 데이터는 오염된 상태로 남게 되고 NoSQL과 같이 구조화된 데이터를 접근하는 프로그램은 오염된 데이터를 읽게 되어 정상적인 서비스가 불가능하게 된다. 더욱 심각한 문제는 이 파일을 원래의 상태로 되돌릴 방법이 없으며, 혹시라도 분산 파일 시스템 자체적으로 제공해 줄 것이라는 기대를 할 수 없다. 왜냐하면 분산 파일 시스템 입장에서는 NoSQL은 수많은 용례 중 하나일 뿐이며 시스템의 가능성을 NoSQL 때문에 축소할 수 없기 때문이다.

본 논문에서는 이러한 문제점을 해결하기 위하여 분산 파일 시스템을 기반으로 하는 NoSQL 구현 시 활용될 수 있는 트랜잭션 기법을 개발하였다.

4. 데이터 신뢰성을 위한 트랜잭션

4.1 완전한 데이터 반영과 성능

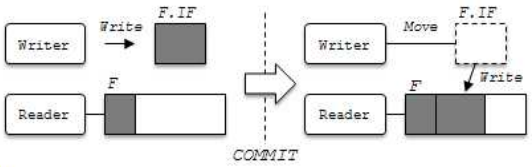
일반적으로 데이터 저장소 시스템에서의 트랜잭션은 데이터 저장 시 무결성을 보장하기 위한 최소의 논리적 작업 단위라는 의미에서 두 가지의 필요성에 의해 사용된다. 첫 번째는 하나의 논리적 작업(예: 삽입, 삭제, 변형)에 대하여 완전한 반영을 하기 위한 단위로써 사용되는 것이고, 나머지는 다수의 사용자, 응용프로그램이 생성하는 복수개의 논리적 작업간의 우선순위 제어 및 상호배제를 관리하기 위한 단위로써 사용된다.

이 두 가지 관점 중 본 논문에서 초점을 맞추는 부분은 분산 파일 시스템과 NoSQL을 연동하였을 경우에 한 트랜잭션 안에서 발생할 수 있는 프로세스의 비정상 종료에 의한 데이터 오염을 방지하는 것이다. 데이터 저장소를 구현하는데 있어 이상적인 트랜잭션 처리는 (1)완전히 반영되거나 전혀 반영되지 않는 “전부 혹은 아무것도”(all-or-nothing)의 원자성 속성을 지켜야만 하며, 이를 위해 지불해야할 (2)성능 측면의 비용이 최대한 작아야만 한다. 본 논문의 방법론은 실상 적대적인 요구조건인 (1), (2)간의 최적의 중간 지점을 찾아 실용적 환경에서 사용할 수 있도록 하는 것을 목표로 하였다.

4.2 접근 방법

앞서 언급한대로, 분산 파일 시스템에서 랜덤 쓰기는 불가능하다. 이것은 파일이 저장되면 되돌릴 수 없다는 것과 동일한 이야기이므로 데이터 오염에 대한 회피가 불가능하다는 것을 알 수 있다. 본 논문에서는 이 문제를 해결하기 위해 중간파일(IF: Inter-File)의 개념을 제안하고자 한다. 중간파일은 트랜잭션에 대한 COMMIT이 이루어질 때까지 임시 파일에 내용을 저장하고 COMMIT이 이루어지는 순간부터 중간파일의 내용을 원본파일로 옮겨 저장하는 작업을 시작 한다. 이 과정에서 COMMIT의 이전 시점에서 프로세스가 종료되면 중간파일은 원본에 적용되지 않았기 때문에 데이터 오염이 방지되고, COMMIT이 이루어진 이후 시점에서는 중간파일로부터 덜 써진 파일의 내용을 이어 쓰는 방식으로 복원한다. 이를 통해 COMMIT 시점을 기준으로 아무것도 저장되지 않거나(nothing), 모든 것이 저장되기(all) 때문에 프로세스의 중도 실패에서 발생할 수 있는 데이터 오염을 방지할 수 있다.

(그림 2) Inter-File을 이용한 데이터 보호



(Figure 2) Data Protection by Inter-File

(그림 2)는 NoSQL이 사용하는 분산 파일 시스템의 논리적 파일 *F*에 대하여 데이터를 쓰고 COMMIT을 통해 데이터를 유효하게 만드는 과정을 보여준다.

그러나 이러한 방식은 성능상의 커다란 문제점을 안고 있다. 중간파일 *F.IF*의 크기는 추정할 수 없으며, 그것이 매우 큰 값일 가능성이 높다. 게다가 옮겨 쓰는 작업이므로 이상적인 상황에서라도 일반 쓰기 작업에 비해 최소한 2배 이상의 시간을 소요한다. 이 문제를 해결하기 위해 분산 파일 시스템의 블록 단위와는 별개로 NoSQL을 위한 고정 길이 블록 구조를 새로 정의 했다.

새로운 NoSQL 블록 구조는 분산 파일 시스템 상에서, 자체적으로 정의한 고정 길이로 파일을 나누어 저장한다. 즉 분산 파일 시스템에서 논리적으로 정의될 수 있는 큰 크기의 파일을 다시 여러 논리적 파일로 나누어 저장하고, 그보다 한 단계 상위의 논리 파일 구조를 구성한다는 것이다. 예를 들어, 128MB의 데이터를 써야하는 상황에서 NoSQL이 64MB 고정길이 블록으로 설정하게 되면, 분산 파일 시스템의 128MB의 논리 파일을 쓰는 대신 64MB의 논리 파일을 두 개 쓴다는 것이다.

이제 다시 중간파일을 쓰는 과정을 생각해보

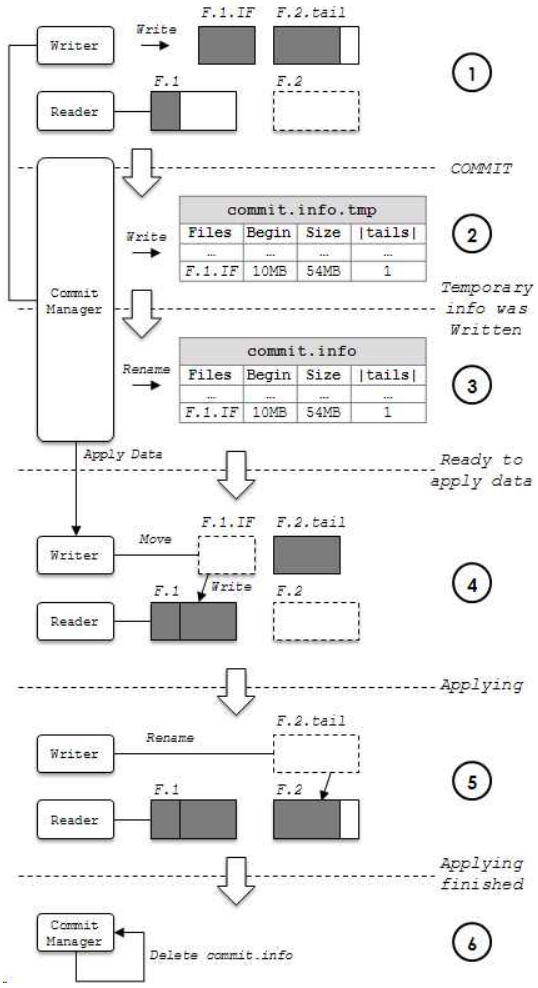
자. 더 이상 중간파일은 매우 큰 크기를 가질 수 있다는 가정을 하지 않아도 된다. 왜냐하면 데이터를 써야하는 첫 번째 블록만 중간파일로 정하고, 그 후의 파일들은 데이터 오염에 걱정이 없으므로 그냥 써도 무방하기 때문이다. 이러한 후위파일의 데이터 오염에 대한 자유성은, 이전 블록이 채워지지 않은 후위파일을 유효성이 없는 파일로 규정함으로써 보장된다.

중간파일의 크기를 규정할 수 있으므로 이론상 데이터 크기의 2배 이상의 쓰기 부하가 드는 원래의 중간파일 방식에 비해 비교할 수 없을 만큼 적은 NoSQL 블록 크기에 비례하는 쓰기 부하가 발생하는 구조로 바뀌어 성능상의 문제를 해결하였다.

(그림 3)은 NoSQL 블록에 데이터를 쓰면서 중간파일 블록 및 후위파일 블록의 정보를 기록하고 COMMIT시 데이터를 적용하는 흐름을 보여준다. ①의 과정에서 NoSQL의 Writer는 중간파일 F.1.IF와 후위파일 F.2.tail에 데이터를 쓴다. COMMIT 명령이 일어나면 ②의 단계에서는 Writer와 연결되어 있는 Commit Manager가 보관하고 있던 중간파일의 쓰기 시작점과 크기를 기록하고 후위파일의 개수를 기록한다. ③의 과정에서는 commit.info.tmp 파일을 commit.info로 변경하는데, 이 시점을 기준으로 파일이 변경되기 이전일 경우에는 모든 쓰기를 무효화하고, 변경된 이후에는 어떤 프로세스가 해당 데이터를 접근하던지에 관계없이 Commit Manager에 의해 나머지 ④,⑤,⑥의 과정을 수행하여 덜 써진 파일을 찾아 이어 쓰기한다.

④에서는 중간파일을 옮겨 쓰는데, 이 과정에서 최대, 블록 크기의 2배의 쓰기 부하가 발생할 수 있지만 이론적으로 무한대의 크기를 가지는 중간파일을 옮겨 쓰는 것에 비할 바는 아니다. ⑤의 과정에서는 단순히 후위파일들을 유효하게 만들기 위해 파일 이름만 변경해준다. 이 과정에서 발생하는 부하는 없다고 보아도 무방하다. 마지막으로 ⑥에서는 commit.info를 지움으로써 정상 상태로 명시한다.

(그림 3) commit 트렌젝션의 흐름도



(Figure 3) Flow of committing transaction

지금까지는 하나의 파일에 대해서만 설명하였다. 그러나 여러 파일에 대하여 한 번의 트랜잭션으로 묶어 관리하는 것도 크게 다르지 않다. commit.info가 써진 이후 시점에서는 어떤 파일을 쓰다가 비정상 종료되더라도 본 논문의 방법론을 사용하는 NoSQL 시스템은 COMMIT된 모든 파일에 대해 이어쓰기 될 것이고, 그렇지 않은 파일들을 무시될 것이다. 그로 인해 모든 데이터는 정상적으로 남지만, 단순히 이 방법만으로는 여러 파일에 대한 쓰기 완료 시점을 맞추는 것은 불가능하다. 그 대신, 각 파일의 유효한 오프셋을 별도로 정의하거나, 각 파일에 정의하는 방법들을 사용할 수 있을 것이며, 이러한 방

법론을 향후 연구의 방향 중 하나로 삼고 있다.

5. 결론 및 향후 연구

고조되는 빅데이터의 관심 속에 안정적이고 확장이 용이한 NoSQL에 대한 요구가 늘어가고 있는 현대에 이르러, 우리는 분산 파일 시스템의 확장성을 계승하는 NoSQL을 구축했다. 이 과정에서 랜덤 쓰기가 금지된 분산 파일 시스템 상에서의 데이터 안정성을 고려하여 중간파일의 개념을 도입하였다. 이를 통해 안전하고 신뢰성 있는 데이터 저장 기능을 제공하는 동시에 사용자가 일관된 데이터를 검색할 수 있도록 하였다. 또한, 이 과정에서 발생할 수 있는 시스템의 성능 측면의 손실을 최소화하려고 노력했다.

본 논문이 두 시스템간의 차이에서 발생한 데이터 안정성의 문제를 물리적, 논리적 파일 수준에서 바라보고 해결책을 모색하는 것에 집중했다면, 향후 연구에서는 여러 파일에 대한 트랜잭션 관리 기법과 분산 파일 시스템의 장점인 Map-Reduce를 활용한 문서 저장 기법에 대해 연구할 것이다.

References

[1] Hilbert, M, López, P., "The World's Technological Capacity to Store, Communicate, and Compute Information," Science 332 (6025): 60 - 65. doi:10.1126/science.1200970, PMID 21310967, 2011.

[2] IBM, "What is big data? - Bringing big data to the enterprise," http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html, Retrieved 2013-08-26.

[3] Sanjay Ghemawat , Howard Gobioff , Shun-Tak Leung, The Google file system, Proceedings of the nineteenth ACM symposium on Operating systems principles, Bolton Landing, NY, USA [doi>10.1145/945445.945450], October 19-22, 2003

[4] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, "The Hadoop Distributed File System," Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), p.1-10, [doi>10.1109/MSST.2010.5496972], May 03-07, 2010.

[5] Jing Han, Haihong, E., Guan Le; Jian Du, "Survey on NoSQL database," Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on , vol., no., pp.363-366, doi: 10.1109/ ICPCA.2011.6106531, 26-28 Oct. 2011.

[6] Younghyun Kwon, Yongseung Kang, Youngmin Ahn, "Business Intelligence Applying Document-Type Bigdata Analysis", Korea information processing society review, vol.19 no.2, pp.86-94, 2012.

[7] L.Sudha Rani, K. Sudhakar , S.Vinay Kumar, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (3) ,3716-3721, 2014.

[8] Sunita Mahajan"Distributed Computing", Oxford University Press.

[9] Jeffrey Dean , Sanjay Ghemawat, MapReduce: simplified data processing on large clusters, Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, p.10-10, San Francisco, CA, December 06-08, 2004.

[10] Apache, "HDFS Architecture Guide," http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, Retrieved 2015-03-06.

[11] Strozzi, Carlo, "NoSQL - A relational database management system," http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page, Retrieved 2015-03-06.

[12] Gajendran, Santhosh Kumar. "A survey on nosql databases." University of Illinois, 2012.

[13] Stonebraker, Michael. "SQL databases v. NoSQL databases," Communications of the ACM 53.4, 10-11, 2010.

[14] Wei, Zhou, Guillaume Pierre, and Chi-Hung Chi, "Cl

oudTPS: Scalable transactions for Web applications in the cloud," Services Computing, IEEE Transactions on 5.4, 525-539, 2012.

[15] Peng, Daniel, and Frank Dabek, "Large-scale Incremental Processing Using Distributed Transactions and Notifications," OSDI. Vol. 10. 2010.

권 영 현

2008년 : 건국대학교 (공학사-인터넷)
2010년 : 건국대학교 대학원 (공학석사-신기술융합 iIT)

2010년~현 재: (주)와이즈넛 과장
관심분야 : 빅데이터 검색 및 분석, 분산 컴퓨팅 등

윤 도 현

2008년 : 가톨릭대학교 (공학사-컴퓨터)

2008년~현 재: (주)와이즈넛 과장
관심분야 : 대용량 데이터 색인 및 복구, 분산 파일 시스템, 추천 시스템 등

박 호 진

2000년 : 한국해양대학교 (공학사-컴퓨터)
2002년 : 한국해양대학교 대학원 (공학석사-컴퓨터공학)

2002년~현 재: (주)와이즈넛 부장 / 팀장
관심분야 : 자연언어처리, 정보검색 등