

하둡 분산 파일 시스템: 구조와 설계

The Hadoop Distributed File System: Architecture and Design

by Dhruba Borthakur

Translated by INHWAN HWANG. <http://www.virgo81.net> / virgo81@gmail.com

목차

1. 소개 (Introduction)	3
2. 가정과 목표 (Assumptions and Goals)	4
2.1. 하드웨어 실패 (Hardware Failure)	4
2.2. 스트리밍 데이터 접근 (Streaming Data Access)	4
2.3. 대용량 데이터 집합 (Large Data Sets)	4
2.4. 단순 간섭 모델 (Simple Coherency Model)	4
2.5. "데이터를 옮기는 것 보다 계산을 옮기는 것이 저렴하다." ("Moving Computation is Cheaper than Moving Data")	4
2.6. 이종 하드웨어와 소프트웨어 플랫폼으로의 이식성 (Portability Across Heterogeneous Hardware and Software Platforms)	5
3. 네임노드와 데이터노드 (Namenode and Datanodes)	6
4. 파일 시스템 네임스페이스 (The File System Namespace)	8
5. 데이터 복제 (Data Replication)	9
5.1. 복제 배치: 걸음마 (Replica Placement: The First Baby Steps)	9
5.2. 복제 선택 (Replica Selection)	10
5.3. 안전모드 (SafeMode)	10
6. 파일 시스템 메타데이터의 영속 (Persistence of File System Metadata)	11
7. 통신 프로토콜 (The Communication Protocols)	12
8. 견고성 (Robustness)	13

8.1. 데이터 디스크 실패, 심장 박동 그리고 재 복제 (Data Disk Failure, Heartbeats and Re-Replication).....	13
8.2. 클러스터 재 균형 (Cluster Rebalancing).....	13
8.3. 데이터 일관성 (Data Integrity).....	13
8.4. 메타데이터 디스크 실패 (Metadata Disk Failure).....	14
8.5. 스냅샷 (Snapshots).....	14
9. 데이터 구성 (Data Organization).....	15
9.1. 데이터 블록 (Data Blocks).....	15
9.2. 옮겨 바꾸기 (Staging).....	15
9.3. 복제 파이프라이닝 (Replication Pipelining).....	15
10. 접근 성.....	17
10.1. DFSShell.....	17
10.2. DFSAdmin.....	17
10.3. 브라우저 인터페이스 (Browser Interface).....	17
11. 공간 교정 (Space Reclamation).....	18
11.1. 파일 삭제 및 복원.....	18
11.2. 복제 인수 감소 (Decrease Replication Factor).....	18
12. 참고문헌 (References).....	19

1. 소개 (Introduction)

하둡 분산 파일 시스템(Hadoop Distributed File System, HDFS)은 기성 하드웨어에서 실행할 수 있도록 디자인된 분산 파일 시스템입니다. 그것은 기존의 분산 파일 시스템과 많은 유사성을 가지고 있습니다.

그러나, 다른 분산 파일 시스템과의 차이점은 매우 큽니다. HDFS는 상당히 내(耐)고장(fault-tolerant)하고 저비용 하드웨어를 통해 배포할 수 있도록 설계되었습니다. HDFS는 응용 프로그램 데이터의 접근에 높은 처리량을 제공하고, 대용량 데이터 집합을 갖는 응용 프로그램에 적합합니다.

HDFS는 파일 시스템 데이터의 스트리밍 접근을 가능하게 하기 위해 약간의 POSIX 요구사항을 게을리하였습니다. HDFS는 애초에 아파치 너치(Apache Nutch) 웹 검색 엔진 프로젝트를 위한 하부 구조로써 만들어졌습니다.

HDFS는 아파치 루씬(Apache Lucene) 프로젝트의 부분인 아파치 하둡(Apache Hadoop) 프로젝트의 부분입니다. 프로젝트의 URL은 <http://projects.apache.org/projects/hadoop.html> 입니다.

2. 가정과 목표 (Assumptions and Goals)

2.1. 하드웨어 실패 (Hardware Failure)

하드웨어 실패는 예외와는 뚜렷하게 다른 기준입니다. HDFS 인스턴스는 파일 시스템의 데이터의 부분을 저장하는 서버 장비의 수백 또는 수천으로 구성될 수 있습니다. 사실 대단히 많은 구성요소가 존재하고 각 구성요소는 불분명한 실패의 확률을 갖습니다. 즉, HDFS의 일부 구성요소는 항상 기능을 수행하지 못합니다. 따라서 결함의 탐지와 빠르고 자동적인 복구는 HDFS의 핵심적인 구조적 목표입니다.

2.2. 스트리밍 데이터 접근 (Streaming Data Access)

HDFS에서 수행되는 응용 프로그램은 그들의 데이터 집합을 위하여 스트리밍 접근을 필요로 합니다. 그들은 일반적으로 일반적인 목적의 파일 시스템에서 수행되는 일반적인 목적의 응용 프로그램이 아닙니다. HDFS는 사용자에게 의한 상호작용적(interactive) 사용 대신 배치 프로세싱에 더 적합하게 설계되었습니다. 강점은 데이터 접근의 낮은 지연(low latency)보다 데이터 접근의 높은 처리입니다. POSIX는 HDFS가 대상으로 하는 응용 프로그램이 필요로 하지 않는 많은 어려운 요구 사항을 부과합니다. 약간의 키 영역들의 POSIX 의미론(semantics)은 데이터 처리율의 향상으로 교환되었습니다.

2.3. 대용량 데이터 집합 (Large Data Sets)

HDFS에서 수행되는 응용 프로그램은 대용량 데이터 집합을 가집니다. HDFS에서의 일반적인 파일은 기가 바이트부터 테라 바이트의 용량입니다. 따라서, HDFS는 대용량 파일들을 지원할 수 있도록 조정(tuned)되었습니다. 그것은 높은 총 데이터 대역폭과 단일 클러스터에서의 수백 노드로의 확장을 제공해야 합니다. 또한 단일 인스턴스에서 수 천만 파일을 제공해야 합니다.

2.4. 단순 간섭 모델 (Simple Coherency Model)

HDFS 응용 프로그램은 파일에 대해 한번 쓰고 여러 번 읽는(write-once-read-many) 접근 모델을 필요로 합니다. 파일이 한번 생성되고, 쓰여지고, 닫히면 변화를 필요로 하지 않습니다. 이런 가정은 데이터 간섭 이슈를 단순화하고 높은 처리량의 데이터 접근을 가능하게 합니다. MapReduce 응용 프로그램 또는 웹 크롤러 응용 프로그램은 이 모델에 완벽하게 적합합니다. 미래에는 파일에 추가-쓰기(appending-writes)를 제공할 계획이 있습니다.

2.5. "데이터를 옮기는 것 보다 계산을 옮기는 것이 저렴하다." ("Moving Computation is Cheaper than Moving Data")

응용 프로그램에 의해 요청된 계산은 만약 그것이 수행하는 데이터 근처에서 실행된다면 보다 더 효율적입니다. 이것은 데이터 집합의 사이즈가 대단히 클 때 특히 사실입니다. 이것은 네트워크

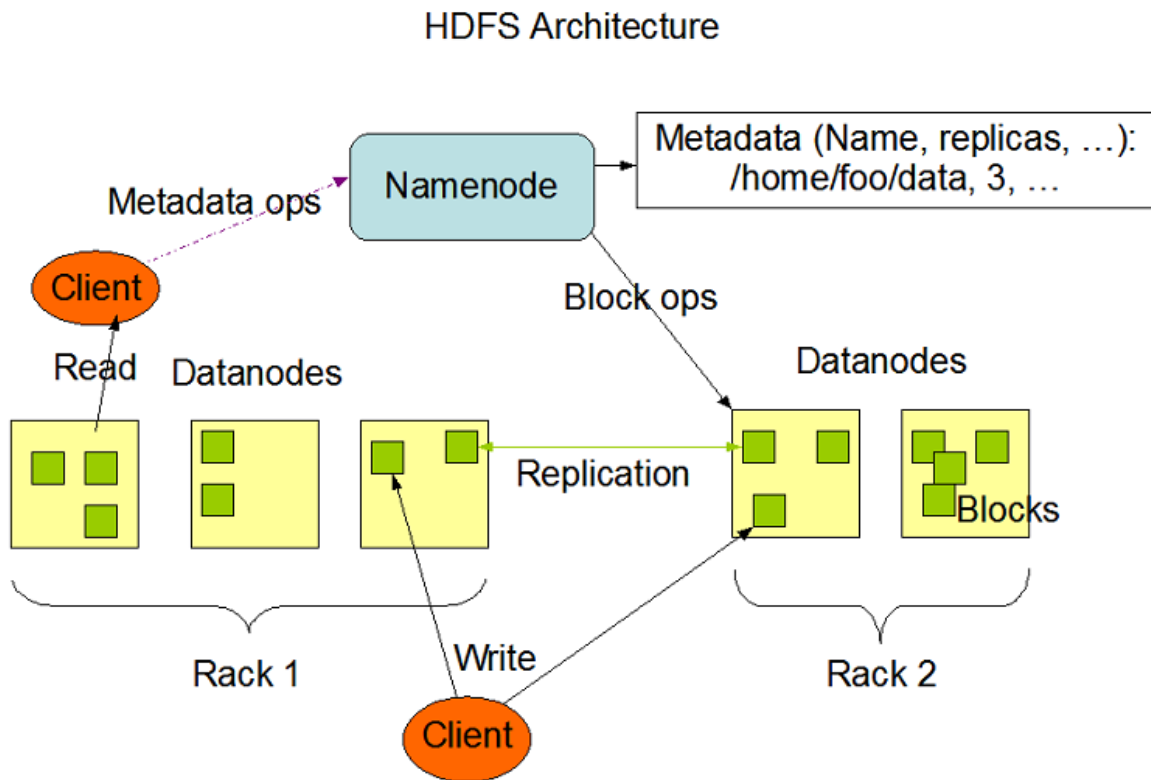
혼잡을 최소화 시키고 시스템의 전체적 처리량을 증가시킵니다. 가정은 응용 프로그램이 수행 되는 곳으로 데이터를 옮기는 것 보다 데이터 가까이로 계산을 옮기는 것이 더 낫다는 것 입니다. HDFS 는 데이터가 위치한 곳 가까이로 그들을 옮길 수 있도록 응용 프로그램을 위한 인터페이스를 제공합니다.

2.6. 이종 하드웨어와 소프트웨어 플랫폼으로의 이식성 (Portability Across Heterogeneous Hardware and Software Platforms)

HDFS 는 한 플랫폼에서 다른 플랫폼으로 쉽게 이식할 수 있도록 디자인되었습니다. 이것은 응용 프로그램의 대형 집합을 위한 선택의 플랫폼으로써 HDFS 의 광범위한 적용을 용이하게 했습니다.

3. 네임노드와 데이터노드 (Namenode and Datanodes)

HDFS 는 주/종 구조를 가지고 있습니다. HDFS 클러스터는 파일 시스템 네임스페이스를 관리하고 클라이언트에 의한 파일 접근을 통제하는 마스터 서버인 단일 *네임노드*로 구성됩니다. 추가적으로 클러스터에서 노드 당 하나며 노드에 붙은 스토리지를 관리하는 수 많은 *데이터노드*가 있습니다. HDFS 는 파일 시스템 네임스페이스를 노출하고 사용자의 데이터를 파일로 저장되도록 허용합니다. 내부적으로 파일은 하나 또는 그 이상의 블록으로 쪼개지며 이러한 블록들은 데이터노드의 집합 안에 저장됩니다. 네임노드는 파일과 디렉터리의 열기, 닫기, 이름 변경과 같은 파일 시스템 네임스페이스 동작을 수행합니다. 또한 데이터노드로의 블록의 매핑을 판단합니다. 데이터노드는 파일 시스템의 클라이언트들로부터의 읽기와 쓰기 요청을 제공하는 역할을 합니다. 또한 네임노드의 지시에 따라 블록 생성, 삭제 그리고 복제를 수행합니다.



네임노드와 데이터노드는 기성 기계에서 실행하도록 설계된 소프트웨어의 조각입니다. 이러한 기계들은 일반적으로 GNU/Linux 운영체제(OS)를 수행합니다. HDFS 는 Java 언어를 이용하여 만들었습니다; Java 를 지원하는 어떤 기계도 네임노드 또는 데이터노드 소프트웨어를 수행할 수 있습니다. 높은 이식성의 Java 언어의 사용은 HDFS 가 광범위한 기계에 배포될 수 있음을 의미합니다. 일반적인 배포는 네임노드만을 수행하는 전용의 기계를 가지고 있습니다. 클러스터에서의 각각의 다른 기계들은 데이터노드 소프트웨어의 한 인스턴스를 수행합니다. 구조는 같은 기계에서의 여러 데이터노드 수행을 제한하지는 않지만 실제 배포에서는 드문 경우

입니다.

클러스터에서의 단일 네임노드의 존재는 시스템의 구조를 대단히 단순화시킵니다. 네임노드는 모든 HDFS 메타데이터를 위한 중재자 그리고 저장소입니다. 시스템은 사용자 *데이터*는 결코 네임노드를 통해 흐르지 않도록 설계되었습니다.

4. 파일 시스템 네임스페이스 (The File System Namespace)

HDFS 전통적인 계층적 파일 구조를 지원합니다. 사용자 또는 응용 프로그램은 디렉터리를 생성할 수 있고 디렉터리에 파일을 저장할 수 있습니다. 파일 시스템 네임스페이스 계층은 대부분의 다른 기존 파일 시스템들과 유사합니다; 파일을 생성하고 지울 수 있고, 하나의 디렉터리에서 다른 디렉터리로 파일을 복사할 수 있고, 파일 이름을 바꿀 수 있습니다. HDFS 에 사용자 한도 량(quotas)과 접근 허가(permissions)는 아직 구현되지 않았습니다. HDFS 는 하드 링크와 소프트 링크를 지원하지 않습니다. 그러나, HDFS 구조는 이러한 특징들의 구현을 제한하지는 않습니다.

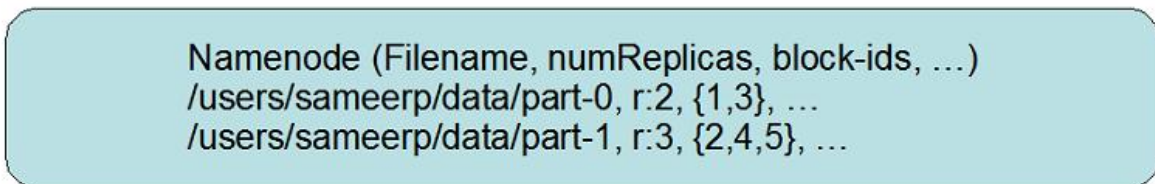
네임노드는 파일 시스템 네임스페이스를 유지합니다. 파일 시스템 네임스페이스 또는 그것의 속성에 가해지는 어떠한 변화도 네임노드에 의해 기록됩니다. 응용 프로그램은 HDFS 에 의해 유지되어야 하는 복제 개수를 명기할 수 있습니다. 파일의 사본 수는 그 파일의 복제 인수라 불립니다. 이러한 정보는 네임노드에 의해 저장됩니다.

5. 데이터 복제 (Data Replication)

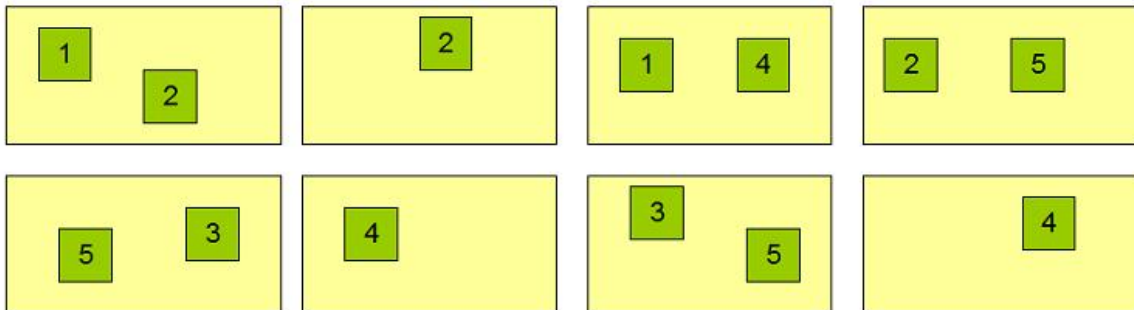
HDFS는 대단위 클러스터에 매우 큰 파일들을 견실히 저장할 수 있도록 설계되었습니다. 그것은 각 파일을 블록의 순서로 저장합니다; 파일에서의 마지막 블록을 제외한 모든 블록은 같은 크기입니다. 파일의 블록들은 내(耐)고장을 위해 복제됩니다. 블록 크기와 복제 인수는 파일마다 설정할 수 있습니다. 응용 프로그램은 파일의 복제의 수를 명기할 수 있습니다. 복제 인수는 파일 생성 시 명기되고 이후에 수정될 수 있습니다. HDFS의 파일은 바꿔 쓰기가 안되고(write-once) 엄격하게 말하면 언제나 하나의 작성자를 갖습니다.

네임스페이스는 블록의 복제에 관하여 모든 결정을 합니다. 그것은 *심장 박동(Heartbeat)*과 *블록 리포트(Blockreport)*를 클러스터 내의 각 데이터노드로부터 주기적으로 받습니다.

Block Replication



Datanodes



5.1. 복제 배치: 걸음마 (Replica Placement: The First Baby Steps)

복제의 배치는 HDFS 신뢰성과 성능에 중요합니다. 복제 배치의 최적화는 HDFS 를 대부분의 다른 분산 파일 시스템과 구분시킵니다. 이것은 많은 조정과 경험이 필요한 특징입니다. 랙을 아는(rack-aware) 복제 배치 정책의 목적은 데이터 신뢰성, 가용성 그리고 네트워크 대역폭 이용을 증대시키는 것 입니다. 복제 배치 정책을 위한 현재의 구현은 이 방향에서의 첫 번째 노력입니다. 이 정책을 구현하는 단기 목표는 상용 시스템에 그것을 검증하고, 그것의 행동에 대해 더 배우고, 그리고 테스트와 연구를 좀 더 정교하게 하는 기반을 만드는 것입니다. 대형 HDFS 인스턴스는 일반적으로 많은 랙에 널리 퍼진 컴퓨터의 클러스터에서 수행합니다. 다른 랙에 있는 두 노드 사이의 통신은 스위치를 통해야만 합니다. 대부분의 경우 같은 랙 안의

기계 간에 네트워크 대역폭은 다른 랙 안의 기계 간의 네트워크 대역폭보다 훨씬 큽니다. 시작할 때, 각 데이터노드는 속해있는 랙을 판단하고 등록 시에 네임노드에게 자신의 랙의 아이디를 알립니다. HDFS 는 기계의 랙 아이디를 판단하는데 사용되는 플러그 할 수 있는(pluggable) 모듈을 용이하게 하기 위한 API 들을 제공합니다. 단순하지만, 최적화되지 않는 정책은 복제를 유일한 랙들에 위치 시키는 것 입니다. 이것은 전체 랙이 실패할 때 데이터를 잃는 것을 방지하고 데이터를 읽을 때 여러 랙으로부터의 대역폭의 사용을 허락합니다. 이 정책은 클러스터 안의 복제를 공평하게 분배합니다. 이 정책은 구성요소 실패 시에도 부하 균형을 쉽게 만들어 줍니다. 그러나 이 정책은 쓰기는 블록을 여러 랙으로 전송해야 하기 때문에 쓰기 비용을 증가 시킵니다.

일반적인 경우, 복제 인수가 3 이고, HDFS 의 배치 정책은 첫 복제를 로컬 랙의 하나의 노드에 놓고, 다른 하나를 로컬 랙의 다른 노드에 놓고 마지막 하나를 다른 랙의 다른 노드에 놓습니다. 이 정책은 랙 사이의 쓰기 트래픽을 감소 시키고 일반적으로 쓰기 성능을 향상시킵니다. 랙 실패의 기회는 노드 실패의 기회보다 훨씬 적습니다; 이 정책은 데이터 신뢰성과 가용성 보장에 영향을 주지 않습니다. 그러나 블록이 유일한 3 개의 랙 대신 2 개의 랙에 위치하기 때문에 데이터를 읽을 때 사용되는 총 네트워크 대역폭은 감소 시킵니다. 이 정책으로 파일의 복제는 랙 사이에 공평하게 분배되지 않습니다. 복제의 1/3 은 한 노드에 있고, 복제의 2/3 은 하나의 랙에 있고 복제의 1/3 은 나머지 랙에 공평하게 분배됩니다. 이 정책은 데이터 신뢰성 또는 읽기 성능의 손상 없이 쓰기 성능을 향상시킵니다.

여기에 설명된 현재의 기본 복제 배치 정책은 진행 중인 작업 입니다.

5.2. 복제 선택 (Replica Selection)

전체 대역폭 소비와 읽기 지연을 최소화하기 위해 HDFS 는 독자로부터 가까운 복제로부터 읽기 요구를 만족 시키는 것을 노력합니다. 만약 독자 노드와 같은 랙에 복제가 존재하면 그 복제가 읽기 요구를 만족 시키기 위해 선호됩니다. 만약 HDFS 클러스터가 여러 데이터 센터로 확장한다면 로컬 데이터 센터에 거주하는 복제가 어떠한 원격 복제보다 선호됩니다.

5.3. 안전모드 (SafeMode)

시작 시, 네임노드는 안전모드라 불리는 특별한 상태에 진입합니다. 데이터 블록의 복제는 네임노드가 안전모드 상태일 때 발생하지 않습니다. 네임노드는 데이터노드로부터 심장 박동(Heartbeat)과 블록리포트 메시지를 받습니다. 블록리포트는 데이터 노드가 호스팅하고 있는 데이터 블록의 목록을 포함합니다. 각 블록은 명기된 복제의 최소 수를 갖고 있습니다. 블록은 네임노드로에 점검된 데이터 블록의 복제의 최소 수 일 때 *안전하게 복제된(safely replicated)* 것으로 고려됩니다. 안전하게 복제된 데이터 블록의 설정 가능한 백분율이 네임노드와 점검한 이후에 (추가적으로 30 초를 더함), 네임노드는 안전모드 상태를 벗어납니다. 그리고 나서 여전히 명기된 수보다 적은 데이터 블록의 목록을 판단합니다. 네임노드는 그리고 나서 이 블록들을 다른 데이터노드에 복제합니다.

6. 파일 시스템 메타데이터의 영속 (Persistence of File System Metadata)

HDFS 네임스페이스는 네임노드에 의해 저장됩니다. 네임노드는 영속적으로 파일 시스템 *메타데이터*에 발생하는 변화를 기록하기 위해 *EditLog*라 불리는 트랜잭션 로그를 사용합니다. 예를 들면 HDFS에서의 새 파일 생성은 네임노드에게 이것을 가리키는 기록을 EditLog에 넣는 것 유발합니다. 비슷하게 파일의 복제 인수의 변경도 EditLog로의 새로운 기록 삽입을 유발합니다. 네임노드는 EditLog를 저장하기 위해 그것의 *지역* 호스트 OS 파일 시스템 안의 파일을 사용합니다. 블록에서 파일로의 매핑과 파일 시스템 속성을 저장하는 전체 파일 시스템 네임스페이스는 *FsImage*라 불리는 파일 안에 저장됩니다. 또한 FsImage는 네임노드의 로컬 파일 시스템 안의 파일로 저장됩니다.

네임노드는 전체 파일 시스템 네임스페이스와 메모리 안의 파일 블록맵의 이미지를 유지합니다. 이 키 메타데이터 항목은 4GB의 램을 가진 네임노드가 거대한 수의 파일과 디렉터리를 지원하기에 충분하도록 작고 경제적으로 설계되었습니다. 네임노드가 시작하면 FsImage와 EditLog를 디스크로부터 읽고 EditLog로부터 모든 트랜잭션을 FsImage의 메모리 내 표현으로 적용하고 이 새 버전을 디스크의 새 FsImage로 내보냅니다. 그리고 나서 그것의 트랜잭션은 영속적으로 FsImage에 적용되었기 때문에 기존 EditLog를 버립니다. 이런 과정을 *체크포인트(checkpoint)*라 부릅니다. 현재 구현에서는 체크포인트는 네임노드가 시작할 때만 발생합니다. 가까운 미래에 주기적으로 체크포인트를 지원하는 작업은 진행 중입니다.

데이터노드는 로컬 파일 시스템의 파일에 HDFS 데이터를 저장합니다. 데이터노드는 HDFS 파일에 관한 지식을 가지고 있지 않습니다. 그것은 그것의 로컬 파일 시스템의 분리된 파일에 HDFS의 각 블록을 저장합니다. 데이터노드는 모든 파일을 같은 디렉터리에 생성하지 않습니다. 대신에 디렉터리 당 파일의 최적화된 수를 판단하고 적절하게 부 디렉터리를 생성하기 위해 휴리스틱(heuristic)을 사용합니다. 같은 디렉터리 내에 모든 로컬 파일을 생성하는 것은 로컬 파일 시스템이 한 디렉터리에 대량의 파일 수를 효과적으로 지원할 수 없기 때문에 최적화가 아닙니다. 데이터노드가 시작할 때 로컬 파일 시스템을 통해 검사하고 로컬 파일 시스템 각각에 상응하는 모든 HDFS 데이터 블록의 목록을 생성하고 네임노드로 보고 합니다: 이것이 블록리포트입니다.

7. 통신 프로토콜 (The Communication Protocols)

모든 HDFS 통신 프로토콜은 TCP/IP 프로토콜 최상에 계층화되어 있습니다. 클라이언트는 네임노드 기계 상의 설정 가능한 TCP 포트로 연결을 설정합니다. 그것은 네임노드와 *클라이언트프로토콜(ClientProtocol)*로 말합니다. 데이터노드는 *데이터노드프로토콜(DatanodProtocol)*을 이용하여 네임노드에게 말합니다. 원격 프로시저 호출(Remote Procedure Call, RPC) 추상은 클라이언트프로토콜과 데이터노드프로토콜로 포장합니다. 설계에 의하면 네임노드는 어떠한 RPC 들도 초기화하지 않습니다. 대신에 그것은 단지 데이터노드와 클라이언트들에 의해 발행된 RPC 요청들만 응답합니다.

8. 견고성 (Robustness)

HDFS의 주된 목표는 실패의 발생에서조차도 데이터를 견실히 저장하는 것입니다. 실패의 세 일반적인 형식은 네임노드 실패, 데이터노드 실패 그리고 네트워크 분할입니다.

8.1. 데이터 디스크 실패, 심장 박동 그리고 재 복제 (Data Disk Failure, Heartbeats and Re-Replication)

각 데이터노드는 네임노드로 심장 박동(Heartbeat) 메시지를 주기적으로 보냅니다. 네트워크 분할은 데이터노드의 부분집합이 네임노드와의 연결을 잃게 하는 것을 유발 시킬 수 있습니다. 네임노드는 심장 박동(Heartbeat) 메시지의 부재를 통해 이런 조건을 발견합니다. 네임노드는 최근 심장 박동(Heartbeat)이 없는 데이터노드를 죽은 것으로 표시하고 새로운 IO 요청을 그들에게 전달하지 않습니다. 죽은 데이터 노드에 등록된 어떤 데이터도 더 이상 HDFS에서 사용 가능하지 않습니다. 데이터노드 죽음은 복제 인수를 어떤 블록의 그들의 명기된 값 이하로 떨어지는 것을 유발합니다. 네임노드는 끊임없이 어떤 블록이 복제되어야 하는지 추적하고 필요할 때 복제를 시작합니다. 재 복제의 필요성은 여러 원인에 의해 발생합니다: 데이터노드가 사용 가능하지 않을 수 있고, 복제가 잘못 될 수도 있고, 데이터노드의 하드디스크가 실패할 수도 있고, 파일의 복제 인수가 증가되었을 수도 있습니다.

8.2. 클러스터 재 균형 (Cluster Rebalancing)

HDFS 구조는 데이터 재 균형 계획(scheme)과 호환 가능합니다. 계획은 만약 데이터노드의 여유 공간이 특정 임계 값 아래로 떨어졌을 때 자동적으로 데이터를 하나의 데이터노드에서 다른 노드로 이동시킬 수 있습니다. 특정 파일에 대한 갑작스런 높은 요구에 계획은 동적으로 추가적인 복제를 생성할 수 있고 클러스터 내의 다른 데이터를 재 균형 시킬 수 있습니다. 데이터 재 균형 계획의 이런 형식은 아직 구현되지 않았습니다.

8.3. 데이터 일관성 (Data Integrity)

데이터노드로부터 가져온 데이터의 블록이 잘못될 수 있습니다. 이러한 변조는 저장 장치의 결함, 네트워크 결함 또는 오류 있는 소프트웨어에 의해 발생할 수 있습니다. HDFS 클라이언트 소프트웨어는 HDFS 파일의 내용을 확인하기 위해 검사 합(checksum)을 구현합니다. 클라이언트가 HDFS 파일을 생성할 때, 파일의 각 블록의 검사 합(checksum)을 계산하고 이 검사 합(checksum)을 같은 HDFS 네임스페이스 안의 분리된 숨겨진 파일로 저장합니다. 클라이언트가 파일 내용을 탐색할 때 연관된 검사 합(checksum) 파일로 저장된 검사 합(checksum)을 데이터와 검증합니다. 만약 아니라면 클라이언트는 그 블록의 복제를 가진 다른 데이터노드로부터 블록을 탐색하는 것을 선택할 수 있습니다.

8.4. 메타데이터 디스크 실패 (Metadata Disk Failure)

FsImage 와 EditLog 는 HDFS 의 중심 데이터 구조입니다. 이러한 파일의 변조는 HDFS 인스턴스를 동작하지 않게 할 수 있습니다. 이러한 이유로 네임노드는 FsImage 와 EditLog 의 여러 사본의 관리를 지원하도록 설정될 수 있습니다. FsImage 또는 EditLog 의 어떠한 갱신도 FsImage 들과 EditLog 들의 각각의 동기적인 갱신을 유발합니다. FsImage 와 EditLog 의 여러 사본의 동기적인 갱신은 네임노드가 지원할 수 있는 초 당 네임스페이스 트랜잭션의 비율을 감소시킬 수 있습니다. 그러나 이러한 감소는 HDFS 태생적으로 응용 프로그램이 *데이터* 집중적이지 *메타데이터* 집중적이지 않기 때문에 수용할만합니다. 네임노드가 재 시작할 때 마지막 일관된 FsImage 와 EditLog 를 사용하기 위해 선택합니다.

네임노드 기계는 HDFS 클러스터에서 실패의 단일 지점입니다. 만약 네임노드 기계가 실패하면 수동적인 조정이 필요합니다. 현재로는 자동적인 재 시작과 다른 기계를 통한 네임노드 소프트웨어의 장애 극복(failover)은 지원되지 않습니다.

8.5. 스냅샷 (Snapshots)

스냅샷은 특정 시점의 순간의 사본의 저장을 지원합니다. 스냅샷 특징의 한 사용은 변조된 HDFS 인스턴스를 이전의 잘 알려진 좋은 시점으로 되돌리는 것일 수 있습니다. HDFS 는 현재 스냅샷을 지원하지 않지만 미래의 발표 물에는 지원할 것 입니다.

9. 데이터 구성 (Data Organization)

9.1. 데이터 블록 (Data Blocks)

HDFS 는 매우 큰 파일을 지원하도록 설계되었습니다. HDFS 와 호환 가능한 응용 프로그램은 큰 데이터 집합을 다루는 그것 입니다. 이러한 응용 프로그램은 그들의 데이터를 한번만 쓰지만 한번 이상 읽고 이러한 읽기가 스트리밍 속도에 만족하는 것을 요구합니다. HDFS 는 파일에서 한번 쓰고 여러 번 읽는 의미론(semantics)을 지원합니다. HDFS 에서 사용되는 일반적인 블록 크기는 64MB 입니다 따라서 HDFS 파일은 64 MB 의 조각으로 잘리고 가능하면 각 조각은 다른 데이터 노드에 존재합니다.

9.2. 옮겨 바꾸기 (Staging)

파일 생성의 클라이언트 요청은 네임노드에 즉각적으로 도달하지 않습니다. 사실, 초기에 HDFS 클라이언트는 파일 데이터를 임시 로컬 파일로 저장합니다. 응용 프로그램의 쓰기는 투명하게 이 임시 로컬 파일로 돌려집니다. 로컬 파일 누적 데이터가 HDFS 의 블록 크기를 넘어서면 클라이언트는 네임노드에 접촉합니다. 네임노드는 파일 이름을 파일 시스템 계층에 삽입하고 그것을 위한 데이터 블록을 할당합니다. 네임노드는 데이터노드와 목적지 데이터 블록의 신원과 함께 클라이언트 요구에 응답합니다. 그리고 나서 클라이언트는 데이터 블록을 로컬 임시 파일로부터 명기된 데이터노드로 내보냅니다. 파일이 닫히면, 임시 로컬 파일 내의 나머지 내보내지지 않은 데이터는 데이터노드로 전송됩니다. 클라이언트는 그리고 나서 네임노드에게 그 파일이 닫혔다고 말합니다. 이 때 네임노드는 파일 생성 동작을 영속 저장소에 넘깁니다. 만약 파일을 닫기 전에 네임노드가 죽으면 파일은 없습니다.

위의 접근법은 HDFS 에서 수행하는 대상 응용 프로그램에 대한 신중한 고려 후에 적용되어야 합니다. 이런 응용 프로그램은 파일로 스트리밍 쓰기를 필요로 합니다. 만약 클라이언트가 클라이언트 측 버퍼링 없이 원격 파일에 직접적으로 쓴다면 네트워크에서 네트워크 속도와 혼잡은 처리량에 상당히 영향을 줍니다. 이러한 접근법은 전례 없는 것은 아닙니다. 초창기 분산 파일 시스템, 예를 들면 AFS 는 성능 향상을 위해 클라이언트 측 캐싱을 사용했습니다. POSIX 요구 사항은 데이터 전송(upload)의 높은 성능을 달성하기 위해 게을리 하였습니다.

9.3. 복제 파이프라이닝 (Replication Pipelining)

클라이언트가 HDFS 파일에 데이터를 쓰는 중일 때, 데이터는 이전에 장에서 설명했던 것처럼 로컬 파일에 먼저 쓰여집니다. HDFS 파일이 세 개의 복제 인수를 가지고 있다고 가정합니다. 로컬 파일이 사용자 데이터의 가득 찬 블록으로 누적 되었을 때 클라이언트는 네임노드로부터 데이터노드의 목록을 탐색합니다. 이 목록은 블록의 복제를 호스트 할 데이터노드를 포함하고 있습니다. 그리고 나서 클라이언트는 데이터 블록을 첫 번째 데이터노드로 내보냅니다. 첫 번째 데이터 노드는 작은 부분(4 KB)으로 데이터를 받기 시작하고 각각의 부분을 로컬 저장소에 쓰고 목록의 두 번째 데이터노드로 전송합니다. 데이터노드의 각 부분을 받기 시작한 두 번째

데이터노드는 부분을 로컬 저장소에 쓰고 세 번째 데이터노드로 내보냅니다. 마지막으로 세 번째 데이터 노드는 데이터를 로컬 저장소에 씁니다. 따라서 데이터노드는 파이프라인 내의 이전 데이터노드로부터 데이터를 받고 동시에 파이프라인 내의 다음 데이터노드로 전송합니다. 따라서 데이터는 한 데이터 노드에서 다음으로 파이프라인으로 보내집니다.

10. 접근 성

HDFS 는 다양한 방법으로 응용 프로그램으로부터 접근될 수 있습니다. 본래, HDFS 는 응용 프로그램이 사용하는 것을 위해 [Java API](#) 를 제공합니다. 이 Java API 를 위한 C 언어 포장(wrapper) 역시 있습니다. 추가적으로 HTTP 브라우저 역시 HDFS 인스턴스의 파일을 브라우저 하는 것에 쓰여집니다. HDFS 를 WebDAV 프로토콜을 통해 노출하는 작업은 진행 중입니다.

10.1. DFSShell

HDFS 는 사용자 데이터를 파일과 디렉터리 형식으로 구성하는 것을 허가합니다. HDFS 내의 데이터와 사용자가 상호작용할 수 있는 *DFSShell* 이라 불리는 명령 줄 인터페이스를 제공합니다. 이 명령 집합의 구문은 사용자가 이미 익숙한 다른 셸(shell)과 유사합니다. (예를 들면 bash, csh). 여기에 약간의 예제 액션/명령 쌍이 있습니다:

액션	명령
/foodir 이름의 디렉터리 생성	bin/hadoop dfs -mkdir /foodir
/foodir/myfile.txt 이름의 파일 내용 보기	bin/hadoop dfs -cat /foodir/myfile.txt

DFSShell 은 저장된 데이터와 상호작용을 위해 스크립팅 언어가 필요한 응용 프로그램을 대상으로 합니다.

10.2. DFSAdmin

DFSAdmin 명령 집합은 HDFS 클러스터를 관리하기 위해 사용됩니다. HDFS 관리자에 의해서만 사용되는 명령들이 있습니다. 여기에 약간의 예제 액션/명령 쌍이 있습니다.

액션	명령
안전모드에서 클러스터 놓기	bin/hadoop dfsadmin -safemode enter
데이터노드의 목록 생성	bin/hadoop dfsadmin -report
datanodname 데이터노드의 중지	bin/hadoop dfsadmin -decommission datanodename

10.3. 브라우저 인터페이스 (Browser Interface)

일반적인 HDFS 설치에는 HDFS 네임스페이스를 설정 가능한 TCP 포트에 노출하기 위해 웹 서버로 설정합니다. 이것은 사용자가 HDFS 네임스페이스를 향해하고 파일의 내용을 웹 브라우저를 통해 보는 것을 허가합니다.

11. 공간 교정 (Space Reclamation)

11.1. 파일 삭제 및 복원

사용자나 응용 프로그램에 의해 파일이 삭제 되었을 때, 그것은 즉각적으로 HDFS 에서 제거되지 않습니다. 대신에 HDFS 는 /trash 디렉터리의 파일로 먼저 이름을 변경합니다. 파일은 /trash 에 남겨진 동안 빠르게 복원 될 수 있습니다. 파일은 /trash 에 설정된 시간 동안 남아있습니다.

/trash 에서의 생활이 완료된 후, 네임노드는 HDFS 네임스페이스에서 그 파일을 지웁니다. 파일의 삭제는 파일과 연계된 블록의 해제를 유발합니다. 사용자에게 의해 지워진 파일과 그에 상응하는 HDFS 에서의 자유 공간 증가는 상당한 지연이 있을 수 있습니다.

사용자는 /trash 디렉터리에 남아있는 동안 삭제 후, 취소할 수 있습니다. 만약 사용자가 지운 파일의 취소를 원한다면 /trash 디렉터리를 향해하고 파일을 탐색할 수 있습니다. /trash 디렉터리는 단지 삭제된 파일의 최근 사본만 포함합니다. /trash 디렉터리는 한가지 특별한 특징(HDFS 는 이 디렉터리로부터 자동적으로 파일을 삭제하기 위해 명기된 정책을 적용할 수 있습니다.)과 함께 어떠한 다른 디렉터리와 같습니다. 현재의 기본 정책은 /trash 로부터 6 시간 이상 된 파일을 지우는 것 입니다. 미래에는 잘 정의된 인터페이스를 통해 정책을 수정할 수 있게 될 것 입니다.

11.2. 복제 인수 감소 (Decrease Replication Factor)

파일의 복제 인수가 감소 되었을 때, 네임노드는 삭제될 수 있는 초과 복제를 선택합니다. 다음 심장 박동(Heartbeat)은 이 정보를 데이터노드로 전송합니다. 데이터노드는 그러면 상응하는 블록을 제거하고 상응하는 자유 공간을 클러스터에 표시합니다. 다시 말하면, setReplication API 호출의 완료와 클러스터 내의 자유 공간 출현에는 시간 지연이 있습니다.

12. 참고문헌 (References)

HDFS Java API: <http://lucene.apache.org/hadoop/api/>

HDFS 소스 코드: http://lucene.apache.org/hadoop/version_control.html