해외 슈퍼컴퓨팅센터의 하둡 동적 실행 기술 분석

2014년 11월

국가슈퍼컴퓨팅연구소

슈퍼컴퓨팅융합연구센터

슈퍼컴퓨팅기술개발실

곽재혁, 황순욱 (KISTI)

KiSTi www.kisti.re.kr 한국과학기술정보연구원 Korea Institute of Science and Technology Information

목 차

표 목 차

그 림 목 차

제 1 장 서론

하둡(Hadoop)은 안정적이고 확장 가능한 분산 컴퓨팅 환경을 위한 오픈 소스 소프트웨어로서 개발되었다. 하둡은 단순한 프로그래밍 모델을 사용하여 컴퓨터 클러스터 상에서 대규모의 데이터를 분산 처리하기 위한 프레임워크를 제공하며 계산 자원과 스토리지를 동시에 제공하는 수천대 규모의 컴퓨터로 확장이 용이하다. 하둡은 어플리케이션 레벨에서 장애를 감지하여 처리할 수 있도록 설계되어 저비용의 컴퓨터 클러스터 상에서 고가용성 서비스를 구축 및 서비스할 수 있다.

하둡은 아파치 프로젝트로 개발되고 있으며 현재 하둡은 다음과 같은 다양한 서브 프로젝트를 포함하고 있다.

- Hadoop Common: 여러 하둡 서브 프로젝트를 지원하는 공통 유틸리티로서 FileSystem, RPC, 직렬화 라이브러리 등을 지원한다.
- Hadoop Distributed File System(HDFS): HDFS는 하둡 응용에 의해서 사용되는 스토리지 시스템으로서 데이터에 대한 고처리 접근성을 제공한다.
- Hadoop MapReduce: Hadoop MapReduce는 컴퓨터 클러스터 상에서 대규모의 데이터를 분산하여 빠르게 처리할 수 있는 프로그래밍 모델과 소프트웨어 프레임워크를 제공한다.

이 외에도 하둡은 다음과 같은 다양한 서브 프로젝트를 포함한다.

- Avro: 데이터 직렬화 시스템
- Cassandra: SPOF(Single Point Of Failure)를 제거한 확장성 있는 멀티-마스터 데이터베이스
- Chukwa: 대규모 분산 시스템을 모니터링하기 위한 데이터 수집 시스템
- HBase: 대용량 테이블에 대한 구조화된 데이터 스토리지를 제공하는 확장성 있는 분산 데이터베이스
- Hive: SQL과 유사한 문법을 이용하여 하둡 데이터에 대한 접근과 분석을 수행할 수 있는 데이터 웨어하우스 시스템
- Mahout: 확장성 있는 기계 학습 및 데이터 마이닝 라이브러리
- Pig: MapReduce 기반 데이터 병렬 처리를 추상화하는 고수준 데이터 흐름 언어 및 실행 환경
- ZooKeeper: 분산 어플리케이션 서비스를 구현하기 위한 설정 정보, 네이밍, 분산 동기화,

그룹 서비스 등을 제공하는 고성능 분산 코디네이터 서비스

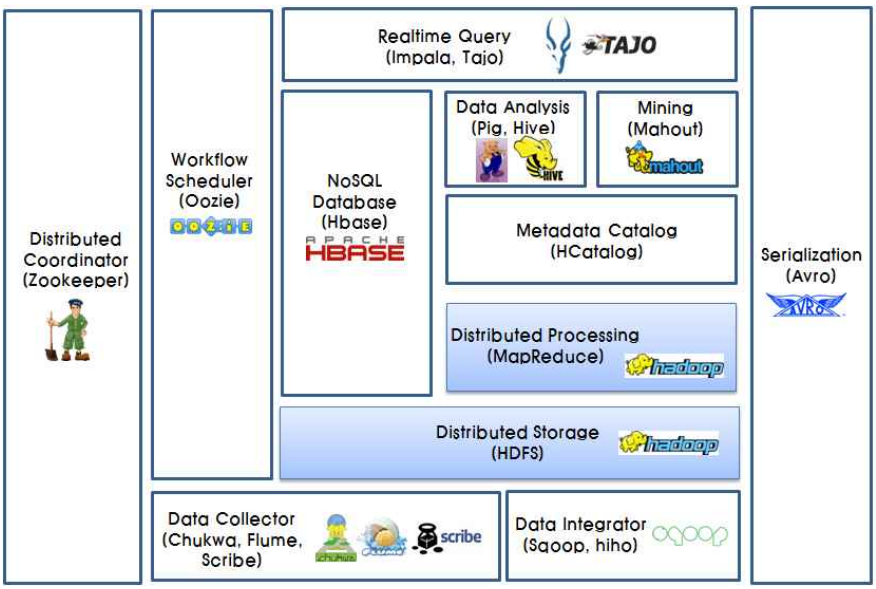

그림 1 Hadoop 에코시스템 현황

하둡은 전용 클러스터를 구축하여 사용하거나 클라우드 상에서 가상 자원을 할당받아 가상화된 하둡 클러스터를 구축하여 사용하는 것이 일반적이다. 여러 행태의 사용자를 대상으로 공유된 자원을 서비스하는 경우에서는 하둡을 사용하는 것이 쉽지 않다. 특히, 이러한 서비스 환경에서는 사용자에게 공정하게 자원을 배분하고 서비스하는 것이 중요한데 하둡은 이런 정책을 고려하여 설계 및 구현되어 있지는 않다.

최근 데이터 분석 위주의 데이터 집약형 응용 연구가 강조됨에 따라서 시뮬레이션 위주의 계산 집약형 응용 연구를 주로 수행했던 슈퍼컴퓨팅센터에서도 하둡을 데이터 집약형 응용 연구를 위한 분산 데이터 처리 프레임워크로서 관심을 보이고 있다. 그러나, 슈퍼컴퓨팅센터에서 서비스되는 자원은 배치큐스케줄러라는 별도의 자원 관리 소프트웨어를 사용하여 사용자에게 공정하게 자원을 할당하고 있으며 전용 클러스터 구축을 전제로 한 하둡을 사용하는 것은 간단한 일이 아니다. 이를 극복하기 위해서 서로 다른 어플리케이션을 가지는 다수의 사용자를 대상으로 서비스하고 있는 공유된 자원 상에서 하둡을 서비스하기 위한 독자적인 소프트웨어 기술을 개발하여 적용하고 있다.

본 보고서에서는 배치큐스케줄러에 의해서 관리되는 클러스터 환경에서 하둡을 동적으로 프로비저닝하여 사용할 수 있는 몇가지 기술들에 대해서 분석하였다. 본 보고서에서는 아파치 커뮤니티 등의 글로벌 오픈소스 커뮤니티와 미국 슈퍼컴퓨팅센터의 사례를 바탕으로 세가지 기술, HOD(Apache), myHadoop(SDSC), SpotHadoop(ORNL)을 선정하였으며 각각의 기술을 분석하고 비교하였다.

제 2 장 하둡

제 1 절 하둡 분산파일시스템

하둡 분산 파일시스템(HDFS)은 아파치 너치 웹검색 엔진을 위한 인프라스트럭쳐로서 개발되었으며 범용성 하드웨어 상에서 동작되도록 설계된 분산 파일 시스템이다. 하둡 분산파일 시스템은 저비용의 하드웨어 상에서 실행되더라도 유연한 장애 처리가 가능하며 대용량 데이터를 처리할 필요가 있는 어플리케이션에 적합하도록 어플리케이션 데이터에 대한 고처리 접근을 제공한다.

하둡 분산파일 시스템을 설계 시에 고려된 가정과 목표를 정리하면 다음과 같다.

- 하드웨어 장애: 하드웨어 장애는 예외적인 상황이 아니라 항상 발생하는 일로 가정하고 있으며 다양한 장애 상황에 대한 탐지와 자동 복구는 하둡 분산파일 시스템의 중요한 목표로 고려되고 있다.
- 스트리밍 데이터 접근 : 하둡 분산 파일 시스템은 데이터에 대한 스트리밍 접근을 필요로 하는 배치 처리 응용에 최적화되어 있으며 데이터 접근에 대한 낮은 지연(low latency)보다는 높은 처리율(high throughput)을 강조한다.
- 대규모 데이터 처리: 하둡분산 파일 시스템은 수 기가바이트에서 수 테라바이트 크기의 데이터를 수백 노드 규모의 단일 클러스터 상에서 고속 분산 처리하는 것을 목표로 하고 있으며 수천만 개의 파일을 저장할 수 있어야 한다.
- 단순한 결합 모델: 하둡 분산파일 시스템의 파일에 대한 WORM(Write-Once-Read-Many) 접근 모델을 필요로 하는 응용 (예를 들면, 로그 분석 혹은 웹 크롤링 등)을 목표로 하고 있다. 이런 특징은 하둡의 결합 모델을 단순화하며 높은 처리율의 데이터 접근을 가능하게 한다.
- 데이터 지역성을 고려한 계산 처리: 하둡은 대규모 데이터 이동시의 네트워크 혼잡을 방지하고 전체 시스템의 처리 효율을 극대화하기 위해서 데이터를 이동하지 않고 응용을 데이터가 위치한 곳으로 이동하여 수행할 수 있는 인터페이스를 제공해야 한다.
- 이기종 하드웨어 및 소프트웨어 호환성: 하둡 분산 파일 시스템은 서로 다른 하드웨어와 소프트웨어 플랫폼 상에서도 잘 동작해야 한다.

하둡 분산파일 시스템은 마스터/슬레이브 구조로 되어 있으며 하나의 네임노드(Namenode)와 다수의 데이터 노드(Datanode)로 구성되어 있다. 네임 노드는 파일시스템 메타데이터를 관리하며 클라이언트로부터 파일에 대한 접근을 제어한다. 데이터 노드는 클러

스터 노드별로 하나씩 실행되며 노드에 연결된 스토리지를 관리한다.

하둡 분산파일 시스템은 내부적으로 파일을 고정된 크기의 블록으로 관리하며 각각의 블록은 데이터 노드에 저장된다. 네임 노드는 파일 및 디렉토리에 대한 열기, 닫기, 이름변경 등의 네임노드 연산을 담당하며 데이터 노드에 저장된 파일을 구성하는 블록에 대한 정보를 관리한다. 데이터 노드는 클라이언트로부터의 읽기/쓰기 요청을 처리하며 네임 노드의 명령을 통해서 블록 생성, 삭제, 복제 등의 블록 연산을 담당한다.

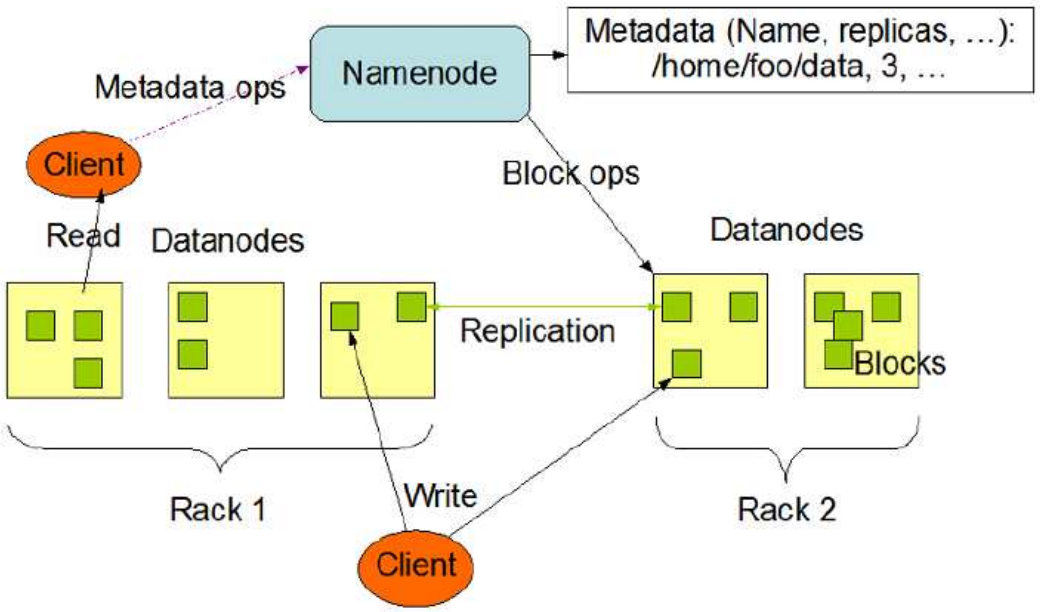

그림 2 하둡 분산파일 시스템의 구조

하둡 분산파일 시스템은 세컨더리네임노드(Secondary NameNode)라는 추가적인 구성 요소를 가지고 있다. 세컨더리네임노드는 하둡 분산파일 시스템의 상태를 모니터링하고 네임노드와 통신하면서 하둡 분산파일 시스템의 스냅샷을 유지한다. 세컨더리네임노드 스냅샷은 네임노드 장애시에 복구 과정에서 이용된다.

제 2 절 하둡 맵리듀스

하둡 맵리듀스는 수천 노드 규모의 범용성 하드웨어 상에서 병렬적으로 수테라바이트 이상의 데이터를 처리하는 프로그램을 작성할 수 있는 소프트웨어 프레임워크이다. 맵리듀스 작업은 맵 태스크에 의해서 병렬 처리되는 독립된 청크(chunk) 단위로 입력 데이터셋를 나눈다. 프레임워크는 맵의 결과를 정렬하고 다시 리듀스 태스크의 입력으로 전달한다. 일반적으로 작업의 입력과 출력은 파일 시스템에 저장된다. 프레임워크는 태스크를 스케줄링하고 모니터링하며 실패한 작업을 재시작하게 된다.

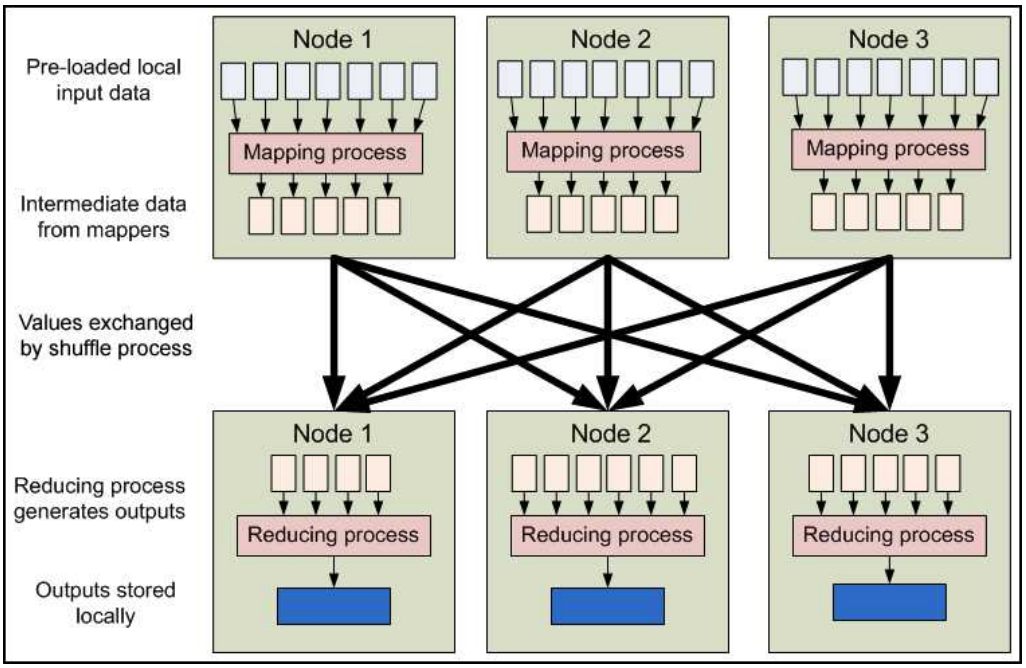

그림 3 맵리듀스 데이터 처리 구조

맵리듀스 프레임워크는 작업의 입력데이터를 <키, 값>의 집합으로 보고 작업의 출력데이터로서 다른 형태의 <키, 값>의 집합을 생성한다. 맵 리듀스의 단계별 키, 값은 다음과 같이 변환되게 된다.

Map(key1, value1) -> list<key2, value2>
Reduce(key2, list<value2>) -> list<value3>

하둡에서는 계산 노드와 스토리지 노드가 동일하며, 즉, 맵리듀스 프레임워크와 하둡 분산파일 시스템은 동일한 노드 상에서 함께 실행된다. 이러한 구성은 데이터가 존재하는 노드에서 태스크를 효율적으로 스케줄링하기 위한 프레임워크를 가능하게 하며 클러스터 상의 대역폭을 최대한 이용할 수 있게 한다.

어플리케이션은 입력 및 출력 데이터 위치를 명시해야 하며, 맵리듀스 프레임워크가 제공하는 인터페이스와 추상 클래스의 구현을 통해서 맵 함수와 리듀스 함수를 제공해야 한다. 작업 클라이언트는 작업(jar실행파일 등)과 작업 설정 정보를 작업 트래커에 제출하는데, 작업 트래커는 계산 노드에서 실행되고 있는 태스크 트래커를 통해서 작업과 작업 설정을 분배하는 역할을 담당하며, 태스크를 스케줄링하고 모니터링하며 작업 클라이언트에게 상태 정보, 분석 정보를 제공하게 된다.

제 3 절 하둡 분산 파일 시스템 명령어 요약

하둡 분산 파일 시스템 쉘은 bin/hadoop fs <args>에 의해서 실행된다. 모든 파일시스템 쉘 명령은 scheme://authority/path 형식의 URI를 전달인자로 받는다. /parent/child와 같은 하둡 분산파일 시스템 상의 파일 혹은 디렉토리는 hdfs://namenodehost/parent/child나 /parent/child로 표현된다. /parent/child로 표현되는 경우 하둡 분산파일 시스템의 설정 파일에 hdfs://namenodehost가 명시되어 있어야 한다.

대부분의 하둡 분산파일시스템 쉘 명령어는 유닉스 명령어와 유사하다. 핵심적인 하둡 분산파일 시스템의 명령어 쉘을 정리하면 다음과 같다.

cat

Usage: hadoop fs -cat URI [URI …]

소스 경로를 표준출력으로 복사한다.

예제:

- hadoop fs -cat hdfs://nn1.example.com/file1 hdfs://nn2.example.com/file2
- hadoop fs -cat file:///file3 /user/hadoop/file4

chgrp

Usage: hadoop fs -chgrp [-R] GROUP URI [URI …]

파일의 그룹 권한을 변경한다. -R이 사용되면 하위 디렉토리까지 모두 변경된다. chgrp는 파일의 소유자거나 슈퍼유저만이 실행 가능하다.

chmod

Usage: hadoop fs -chmod [-R] <MODE[,MODE]... | OCTALMODE> URI [URI …]

파일의 접근 권한을 변경한다. -R이 사용되면 하위 디렉토리까지 모두 변경된다. chmod는 파일의 소유자거나 슈퍼유저만이 실행 가능하다.

chown

Usage: hadoop fs -chown [-R] [OWNER][:[GROUP]] URI [URI]

파일의 소유자 권한을 변경한다. -R이 사용되면 하위 디렉토리까지 모두 변경된다.

chmod는 파일의 소유자거나 슈퍼유저만이 실행 가능하다.

copyFromLocal

Usage: hadoop fs -copyFromLocal <localsrc> URI

소스가 로컬 파일 경로인 것을 제외하고 put과 동일하다.

copyToLocal

Usage: hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>

목적지가 로컬 파일 경로인 것을 제외하고 get과 동일하다.

cp

Usage: hadoop fs -cp URI [URI …] <dest>

소스에서 목적지로 파일을 복사한다. 목적지가 디렉토리인 경우 복수개의 소스를 지정할 수 있다.

예제:

- hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2
- hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2 /user/hadoop/dir

get

Usage: hadoop fs -get [-ignorecrc] [-crc] <src> <localdst>

하둡 분산파일 시스템에서 로컬 파일 시스템으로 파일을 복사한다.

예제:

- hadoop fs -get /user/hadoop/file localfile
- hadoop fs -get hdfs://nn.example.com/user/hadoop/file localfile

getmerge

Usage: hadoop fs -getmerge <src> <localdst> [addnl]

소스 디렉토리 안에 파일들을 병합하여 목적지 로컬 파일로 내보낸다.

ls

Usage: hadoop fs -ls <args>

파일인 경우, 다음의 형식을 따르는 파일 정보를 보여준다.

permissions number_of_replicas userid groupid filesize modification_date modification_time filename

디렉토리인 경우, 다음의 형식을 따르는 디렉토리 하위 목록을 보여준다.

permissions userid groupid modification_date modification_time dirname

예제:

hadoop fs -ls /user/hadoop/file1

mkdir

Usage: hadoop fs -mkdir <paths>

경로에 해당하는 디렉토리를 상위 경로를 포함하여 생성한다.

예제:

- hadoop fs -mkdir /user/hadoop/dir1 /user/hadoop/dir2
- hadoop fs -mkdir hdfs://nn1.example.com/user/hadoop/dir hdfs://nn2.example.com/user/hadoop/dir

mv

Usage: hadoop fs -mv URI [URI …] <dest>

소스에서 목적지로 파일을 이동한다. 목적지가 디렉토리인 경우 복수개의 소스 파일이 지정될 수 있다. 파일 시스템간 파일 이동은 지원되지 않는다.

예제:

- hadoop fs -mv /user/hadoop/file1 /user/hadoop/file2
- hadoop fs -mv hdfs://nn.example.com/file1 hdfs://nn.example.com/file2 hdfs://nn.example.com/file3 hdfs://nn.example.com/dir1

put

Usage: hadoop fs -put <localsrc> ... <dst>

하나 이상의 소스 파일 경로를 목적지 파일 시스템으로 복사한다. 또한 표준 입력으로 읽어서 목적지 파일 시스템으로 쓸 수도 있다.

예제:

- hadoop fs -put localfile /user/hadoop/hadoopfile
- hadoop fs -put localfile1 localfile2 /user/hadoop/hadoopdir
- hadoop fs -put localfile hdfs://nn.example.com/hadoop/hadoopfile
- hadoop fs -put - hdfs://nn.example.com/hadoop/hadoopfile

rm

Usage: hadoop fs -rm [-skipTrash] URI [URI …]

파일이나 디렉토리를 삭제한다. 삭제는 비어있는 디렉토리와 파일만 가능하다.

예제:

- hadoop fs -rm hdfs://nn.example.com/file /user/hadoop/emptydir

rmr

Usage: hadoop fs -rmr [-skipTrash] URI [URI …]

하위 경로를 포함하여 파일이나 디렉토리를 삭제한다.

예제:

- hadoop fs -rmr /user/hadoop/dir
- hadoop fs -rmr hdfs://nn.example.com/user/hadoop/dir

setrep

Usage: hadoop fs -setrep [-R] <path>

파일의 복제 개수를 변경한다.

예제:

- hadoop fs -setrep -w 3 -R /user/hadoop/dir1

stat

Usage: hadoop fs -stat URI [URI …]

경로에 대한 상태 정보를 보여준다.

예제:

- hadoop fs -stat path

제 3 장 Apache HOD

제 1 절 HOD 소개

HOD(Hadoop On Demand)는 공유된 클러스터 노드 상에서 독립적인 하둡 인스턴스를 실행할 수 있는 시스템으로 아파치 오픈소스 프로젝트로 개발되었다. HOD는 관리자나 사용자가 하둡을 셋업하고 사용할 수 있는 도구를 제공하며, 하둡 개발자나 테스터가 공유된 물리 클러스터 환경에서 자신만의 하둡 버전을 테스트하는 것을 가능하게 한다.

HOD는 물리 클러스터 상에 노드 할당을 위하여 Torque를 사용한다. 할당된 노드 상에서 하둡 맵리듀스나 HDFS 데몬이 실행될 수 있으며 하둡 데몬이나 클라언트를 위한 설정 파일(hadoop-site.xml 등)을 자동으로 생성한다.

HOD의 특징을 정리하면 다음과 같다.

- 하둡 클러스터 프로비저닝 및 관리

HOD의 가장 큰 특징은 하둡 맵리듀스와 HDFS로 구성된 하둡 클러스터를 동적으로 프로비저닝할 수 있다는 것이다. 가용한 노드가 있고 운영 정책이 허용된다면 사용자는 동시에 여러개의 하둡 클러스터를 할당하기 위해서 HOD를 사용할 수 있다.

- 하둡을 분배하기 위해서 tarball을 사용

하둡을 프로비저닝할 때 HOD는 클러스터 상에 미리 설치된 하둡을 사용할 수도 있고 프로비저닝의 일부로서 하둡 tarball을 설치할 수도 있다. 이것은 하둡 개발자가 공유 클러스터 상에서 하둡을 테스트할 목적으로 여러 버전의 하둡을 사용하는 것을 가능하게 한다.

- 외부 HDFS의 사용

HOD에 의해서 하둡 클러스터가 프로비저닝되는 경우에 정적으로 미리 설정되어 있는 HDFS를 사용할 수 있다. 이것은 HOD에 의해서 실행된 하둡 클러스터의 할당이 종료된 후에도 데이터를 유지할 수 있는 것을 허락한다.

- 하둡을 설정하기 위한 다양한 옵션 가능

HOD는 HOD 설정 파일이나 HOD 클라이언트의 명령어 옵션을 통해서 하둡 데몬과 하둡 설정 파일을 구성하기 위한 편리한 방법을 제공한다.

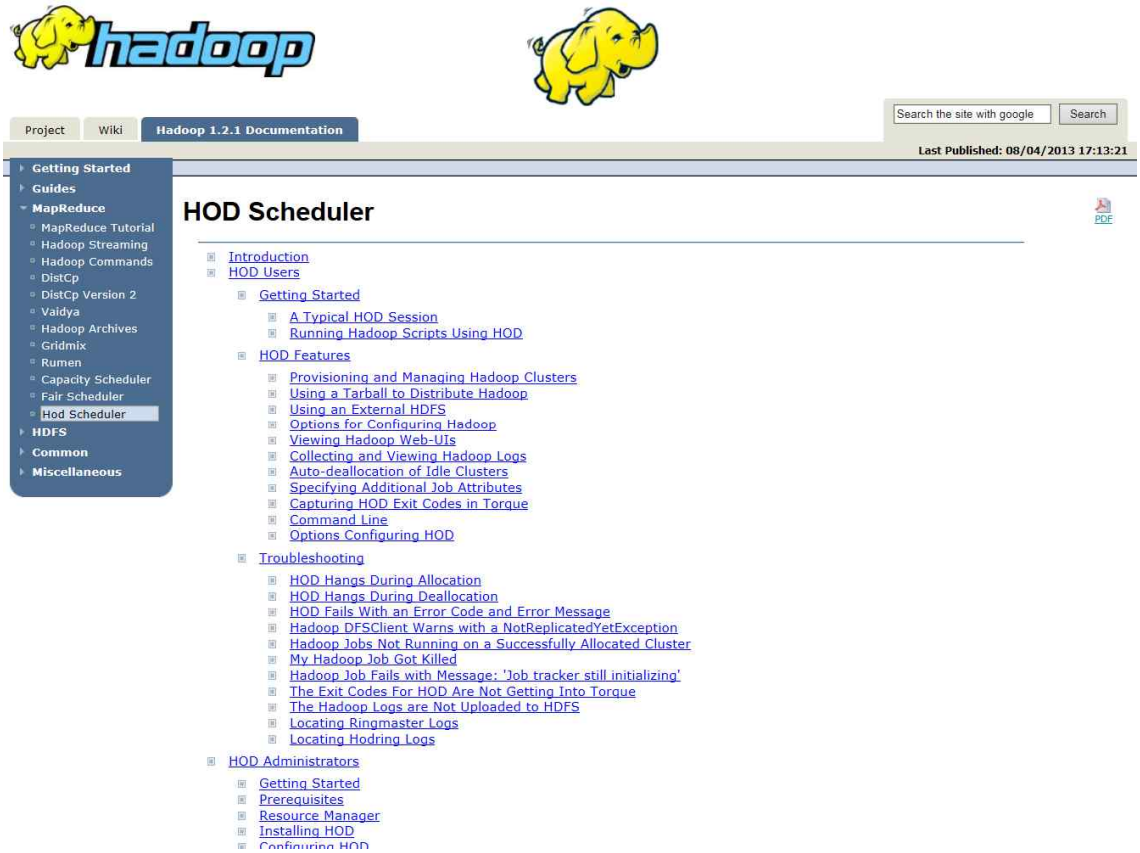
Project
Wiki
Hadoop 1.2.1 Documentation
Search the site with google
Search
Last Published: 08/04/2013 17:13:21
Getting Started
Guides
MapReduce
MapReduce Tutorial
Hadoop Streaming
Hadoop Commands
DistCp
DistCp Version 2
Vaidya
Hadoop Archives
Gridmix
Rumen
Capacity Scheduler
Fair Scheduler
Hod Scheduler
HDFS
Common
Miscellaneous
HOD Scheduler
PDF
Introduction
HOD Users
Getting Started
A Typical HOD Session
Running Hadoop Scripts Using HOD
HOD Features
Provisioning and Managing Hadoop Clusters
Using a Tarball to Distribute Hadoop
Using an External HDFS
Options for Configuring Hadoop
Viewing Hadoop Web-UIs
Collecting and Viewing Hadoop Logs
Auto-deallocation of Idle Clusters
Specifying Additional Job Attributes
Capturing HOD Exit Codes in Torque
Command Line
Options Configuring HOD
Troubleshooting
HOD Hangs During Allocation
HOD Hangs During Deallocation
HOD Fails With an Error Code and Error Message
Hadoop DFSClient Warns with a NotReplicatedYetException
Hadoop Jobs Not Running on a Successfully Allocated Cluster
My Hadoop Job Got Killed
Hadoop Job Fails with Message: 'Job tracker still initializing'
The Exit Codes For HOD Are Not Getting Into Torque
The Hadoop Logs are Not Uploaded to HDFS
Locating Ringmaster Logs
Locating Hodring Logs
HOD Administrators
Getting Started
Prerequisites
Resource Manager
Installing HOD
Configuring HOD

그림 4 HOD 웹사이트

제 2 절 HOD의 내부 구조

HOD의 기본적인 시스템 아키텍쳐는 (1) 자원관리자 및 스케줄러(RM), (2) RM과 통신하는 HOD 컴포넌트인 RingMaster와 HODRing, (3) 이를 통해서 실행되는 하둡 맵리듀스 및 HDFS 데몬, 그리고, (4) 클라이언트 구성요소로서 하둡 클라이언트와 HOD 쉘로 구분할 수 있다.

HOD의 사용자 세션은 크게 클러스터 상의 노드 할당, 노드 상에서 하둡 작업 실행, 클러스터 노드 할당 해제의 세가지 단계로 구성된다. 다음은 HOD의 사용자 세션에 대해서 간략하게 기술하면 다음과 같다.

- 사용자는 HOD쉘을 사용하여 Torque에 필요한 규모의 컴퓨팅 노드를 요청한다. 사용자 요청에는 노드의 수, RM에 의해서 시작될 RingMaster라고 불리는 특별한 프로세스에 대한 설명, 하둡 설치에 대한 명세를 포함한다.
- Torque는 충분한 노드가 가용할때까지 사용자 요청을 큐에 둔다. 일단, 노드가 가용하게 되면, Torque는 RingMaster라 불리는 특별한 프로세스를 할당된 노드의 하나에서 시작한다.
- RingMaster는 HOD 컴포넌트 중 하나이며, RingMaster는 RM 인터페이스를 사용하여 할당된 각각의 노드에서 실행될 HODRing이라는 또다른 HOD 컴포넌트를 실행한다.
- HODRing이 시작되면, RingMaster와 통신하여 하둡 명령어를 가져와서 실행한다. 일단, 하둡 데몬이 시작되면, HODRing은 RingMaster에 등록되고 하둡 데몬에 관한 정보를 제공한다.
- HOD 클라이언트는 잡트래커와 HDFS 데몬의 위치를 알기 위해서 RingMaster와의 통신을 유지한다.
- 모든 것이 셋업되고 사용자가 잡트래커와 HDFS 데몬의 위치를 알게되면 HOD는 사용자가 할당된 클러스터 상에서 하둡 기반의 데이터 분석을 수행하도록 허락한다.
- 사용자는 일단 데이터 분석 작업이 끝나면 할당된 클러스터를 릴리즈한다.

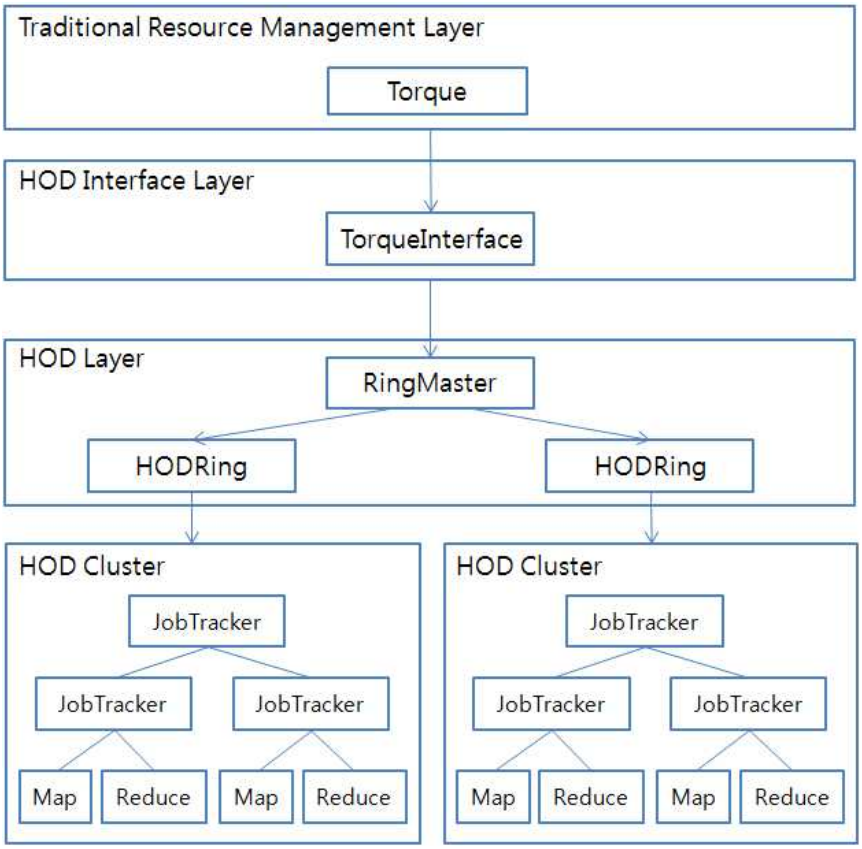
Traditional Resource Management Layer
Torque
HOD Interface Layer
TorqueInterface
HOD Layer
RingMaster
HODRing
HODRing
HOD Cluster
JobTracker
JobTracker
JobTracker
Map
Reduce
Map
Reduce
HOD Cluster
JobTracker
JobTracker
JobTracker
Map
Reduce
Map
Reduce

그림 5 HOD 아키텍처

제 3 절 HOD의 설치 및 테스트

여기서는 리눅스 기반으로 HOD를 설치하고 테스트하는 방법을 설명한다. HOD는 리눅스 기반으로 설치될 수 있으며 HOD가 설치된 클러스터에는 RM으로서 Torque를 사용하기 때문에 Torque가 미리 설치되어 있어야 하며 Python으로 구현되어 있으므로 Python 또한 설치가 필요하다. 또한 HOD를 사용하기 위해서는 최소 3개의 계산 노드가 필요하다.

Torque는 http://www.adaptivecomputing.com에서 다운로드 가능하다. Torque에서 pbs_server는 클러스터의 헤드 노드에 설치되며, pbs_mom은 모든 계산노드에 설치되어 실행하고 있어야 한다. pbs_server에 작업을 제출할 큐를 생성해야 하는데 큐의 이름은 HOD 설정 파라메터인 resource-manager.queue와 동일한 이름을 가져야 한다. HOD 클라이언트는 RingMaster 프로세스를 Torque 작업으로서 큐에 제출한다. 클러스터 노드를 위한 프로퍼티로서 클러스터명을 설정해야 하는데 qmgr -c "set node node properties=cluster-name"으로 설정 가능하다. 클러스터명은 HOD 설정 파라메터인 hod.cluster와 동일해야 한다. Torque의 제대로 설치되었는지를 확인하기 위해서 echo "sleep 30" | qsub -l nodes=3과 같은 간단한 작업을 통해서 테스트할 수 있다.

Torque가 설치되었으면 HOD를 설치할 차례이다. HOD는 하둡 tarball의 contrib 디렉토리 아래에 hod라는 디렉토리 안에 포함되어 있다. hod 디렉토리는 클러스터 상의 모든 노드에서 공유되는 위치로 복사한다.

HOD를 실행하기를 원하는 노드에서 HOD 설정 파일인 hodrc를 수정한다. hodrc는 <install dir>/conf 디렉토리에서 찾을 수 있다. HOD 설정 파일은 HOD를 실행하기 위한 최소한의 키-값을 포함하고 있다.

다음과 같은 값들이 실제 클러스터 환경에 맞게 수정되어야 한다.

${JAVA_HOME}: JDK 1.6.1이상의 자바가 설치된 위치

${CLUSTER_NAME}: RM의 노드 프로퍼티 설정에서 명시한 클러스터명

${HADOOP_HOME}: 하둡이 설치된 위치

${RM_QUEUE}: HOD 작업이 제출될 RM의 큐 이름

${RM_HOME}: RM이 설치된 위치

다음은 hodrc의 예를 보인 것이다.

```
$ cat hodrc
[hod]
```

```
stream                          = True
java-home                       = /opt/jdk1.6.0_43
cluster                         = er001.ksc.re.kr
cluster-factor                  = 1.8
xrs-port-range                  = 32768-65536
debug                           = 3
allocate-wait-time              = 3600
temp-dir                        = /tmp/hod

[ringmaster]
register                        = True
stream                          = False
temp-dir                        = /tmp/hod
http-port-range                 = 8000-9000
work-dirs                       = /tmp/hod/1,/tmp/hod/2
xrs-port-range                  = 32768-65536
debug                           = 3

[hodring]
stream                          = False
temp-dir                        = /tmp/hod
register                        = True
java-home                       = /opt/jdk1.6.0_43
http-port-range                 = 8000-9000
xrs-port-range                  = 32768-65536
hadoop-port-range               = 50000-60000
debug                           = 3

[resource_manager]
```

```
queue                          = batch
batch-home                     = /opt/torque-4.2.6.1
id                             = torque
#env-vars                      = HOD_PYTHON_HOME=/foo/bar/python-2.5.1/bin/python

[gridservice-mapred]
external                       = False
pkgs                           = /home/jhkwak/hadoop-stack/hadoop-1.2.1
tracker_port                   = 8030
info_port                      = 50080

[gridservice-hdfs]
external                       = False
pkgs                           = /home/jhkwak/hadoop-stack/hadoop-1.2.1
fs_port                        = 8020
info_port                      = 50070
```

다음은 HOD 클라이언트를 이용하여 HOD를 실행할 차례이다. 먼저 클러스터 디렉토리를 생성한다. 클러스터 디렉토리는 HOD가 하둡 관련 설정 파일인 hadoop-site.xml을 자동으로 생성하여 저장하는 디렉토리이다. 클러스터 디렉토리가 없을 경우 HOD는 자동으로 생성하여 사용한다. 여기서는 ~/hadoop-cluster/라는 디렉토리로 가정한다.

HOD가 동적으로 셋업할 하둡 tarball을 모든 노드에서 공유되는 위치에 복사해둔다. 여기서는 ~/shared/라는 디렉토리로 가정한다.

HOD 설정 파일인 hodrc의 위치를 지정하기 위해서 HOD_CONF_DIR 환경 변수를 설정한다. 여기서는 ~/hod-config로 가정한다.

```
$ export HOD_CONF_DIR=~/hod-config
```

HOD를 통해 클러스터 노드를 할당하기 위해서 다음과 같이 실행한다.

```
$ hod -c ~/hod-config/hodrc -t ~/shared/hadoop-1.2.1-bin.tar.gz -o "allocate ~/hadoop-cluster 3"
```

/home/jhkwak/hadoop-stack/hod/hodlib/Common/desc.py:21: DeprecationWarning: the sets module is deprecated

from sets import Set

INFO - Cluster Id 83.er001.ksc.re.kr

INFO - HDFS UI at http://er002.ksc.re.kr:58467

INFO - Mapred UI at http://er003.ksc.re.kr:54274

INFO - hadoop-site.xml at /home/jhkwak/hadoop-cluster

HOD 클라이언트의 실행 결과를 보면 er002가 네임노드로 실행되며 er003이 잡트래커 노드로 선택되었음을 알 수 있다.

Torque의 qstat 명령을 통해서 HOD 작업에 대해서 확인 가능하다.

$ qstat -an

er001.ksc.re.kr:

Req'd Req'd Elap

Job ID Username Queue Jobname SessID NDS TSK Memory Time S Time

---

97.er001.ksc.re.kr jhkwak batch HOD 29244 3 3 -- 01:00:00 R 00:18:41

er002+er004+er003

qstat의 실행 결과를 보면 er002, er003, er004가 사용됨을 알 수 있으며 위의 정보가 종합해볼 때 er004가 데이터노드 및 태스크트래커 노드로 사용됨을 확인할 수 있다. 이것은 jps 명령어를 통해서도 각 노드에 실행된 하둡 프로세스를 확인해 볼수 있다.

다음은 HOD에서 셋업된 하둡 클러스터를 테스트할 차례이다. 여기서는 wordcount라는 예제를 실행할 것이다. 먼저 하둡 클러스터 상에 입력데이터를 저장하고 있는 디렉토리를

생성하고 입력데이터를 전송한다. 그리고, 하둡 명령을 통해서 맵리듀스 작업을 수행하고 결과 디렉토리에서 결과데이터를 확인한다. 하둡 명령어를 실행할 때 -c옵션을 통해서 하둡 설정 파일의 위치를 지정하여야 하는데 앞서 하둡 설정 파일이 생성되는 위치인 클러스터 디렉토리를 ~/hadoop-cluster로 지정한 바 있다. 다음은 wordcount 예제를 실행한 예이다.

```
$ hadoop --config ~/hadoop-cluster dfs -mkdir data
$ hadoop --config ~/hadoop-cluster dfs -put ./pg2701.txt data/
$ hadoop --config ~/hadoop-cluster dfs -ls data
-rw-r--r--  3 jhkwak supergroup  1257274 2014-03-14 16:54 /user/jhkwak/data/pg2701.txt
$ hadoop --config ~/hadoop-cluster jar ~/hadoop-stack/hadoop-1.2.1/hadoop-examples-1.2.1.jar wordcount data wordcount-output
$ hadoop --config ~/hadoop-cluster dfs -ls wordcount-output
-rw-r--r--  3 jhkwak supergroup  0 2014-03-14 16:59 /user/jhkwak/wordcount-output/_SUCCESS
drwxr-xr-x  - jhkwak supergroup  0 2014-03-14 16:58 /user/jhkwak/wordcount-output/_logs
-rw-r--r--  3 jhkwak supergroup  123211 2014-03-14 16:58 /user/jhkwak/wordcount-output/part-r-00000
-rw-r--r--  3 jhkwak supergroup  121628 2014-03-14 16:59 /user/jhkwak/wordcount-output/part-r-00001
-rw-r--r--  3 jhkwak supergroup  121827 2014-03-14 16:59 /user/jhkwak/wordcount-output/part-r-00002
$ hadoop --config ~/hadoop-cluster dfs -get wordcount-output ./
```

HOD를 통해서 할당된 노드를 해제하기 위해서는 다음과 같이 실행하면 된다.

```
$ hod -c ~/hod-config/hodrc -o "deallocate ~/hadoop-cluster"
```

제 4 절 University of Chicago의 HOD 활용 사례

University of Chicago 대학은 HOD를 이용해서 하둡을 서비스하고 있다. 다음 그림은 University of Chicago 대학의 HOD를 통한 하둡 작업을 실행하는 방법을 설명한 웹사이트를 보인 것이다.

그림 6 University of Chicago의 하둡 서비스 관련 사이트

제 4 장 myHadoop

제 1 절 myHadoop 소개

Hadoop과 같은 Shared Nothing을 구조를 가지는 시스템 아키텍처는 자원이 공유된 형태로 서비스되는 HPC 자원과는 차이가 있다. 다음 그림은 두가지 시스템 아키텍처를 도식화한 것이다. 가장 큰 차이점으로는 HPC 환경에서는 인피니밴드와 같은 고속 네트워크를 사용하는 병렬 파일시스템 기반의 공유스토리지를 사용하며 계산 노드에서는 최소한의 로컬 스토리지만을 사용하는 반면, 하둡과 같은 Shared Nothing 환경에서는 공유된 스토리지를 사용하지 않고 각 계산 노드의 로컬 스토리지를 활용한다는 것이다. 또한, HPC 환경에서는 계산 노드에 대한 접근을 위해서 Torque, SGE와 같은 배치 시스템을 활용하는 반면, Shared Nothing 환경에서는 계산노드에 직접적으로 접근하고 계산노드와 데이터노드가 함께 구성된다. 따라서 하둡 작업을 공유된 HPC 자원에서 실행하기 위해서는 네이티브 배치 스케줄러 환경에서 수행될 수 있어야 하고 HPC 환경 위에서 Shared Nothing 환경을 구현할 수 있어야 한다.

myHadoop은 HPC 자원 위에서 하둡을 온디맨드 방식으로 실행할 수 있는 시스템으로 미국 샌디에고 슈퍼컴퓨팅센터에서 개발되었다. myHadoop을 이용하여 사용자는 HPC 환경에서 하둡 코드를 개발하고 실행하는 것이 가능해졌다. myHadoop은 https://github.com/glennklockwood/myhadoop/에서 다운로드 가능하다.

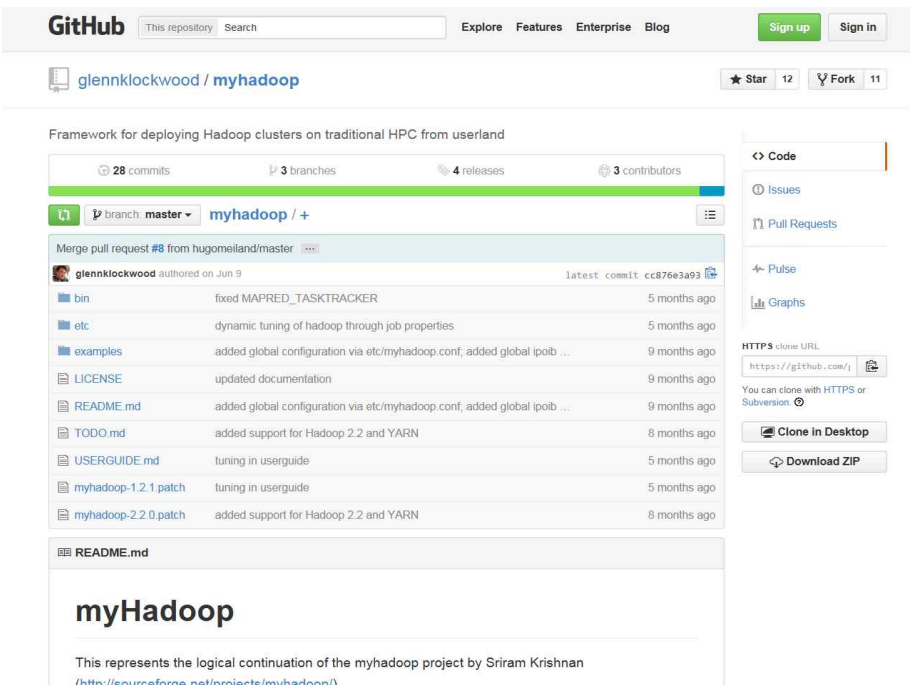

그림 7 myHadoop 웹사이트

제 2 절 myHadoop의 내부 구조

다음 그림은 HPC 환경에서 myHadoop의 전체적인 아키텍처를 도식화한 것이다. 사용자는 myHadoop을 이용하여 배치스케줄러 기반의 HPC 자원 상에서 하둡 클러스터를 온디맨드로 할당할 수 있다. myHadoop은 배치스케줄러를 통해서 마스터 노드와 슬레이브 노드를 할당받고 할당된 노드 위에서 하둡 데몬을 실행하여 하둡 클러스터를 구성하고 사용자는 하둡 클러스터 상에서 하둡 작업을 수행할 수 있으며 할당된 노드를 해제함으로서 하둡 클러스터를 종료한다.

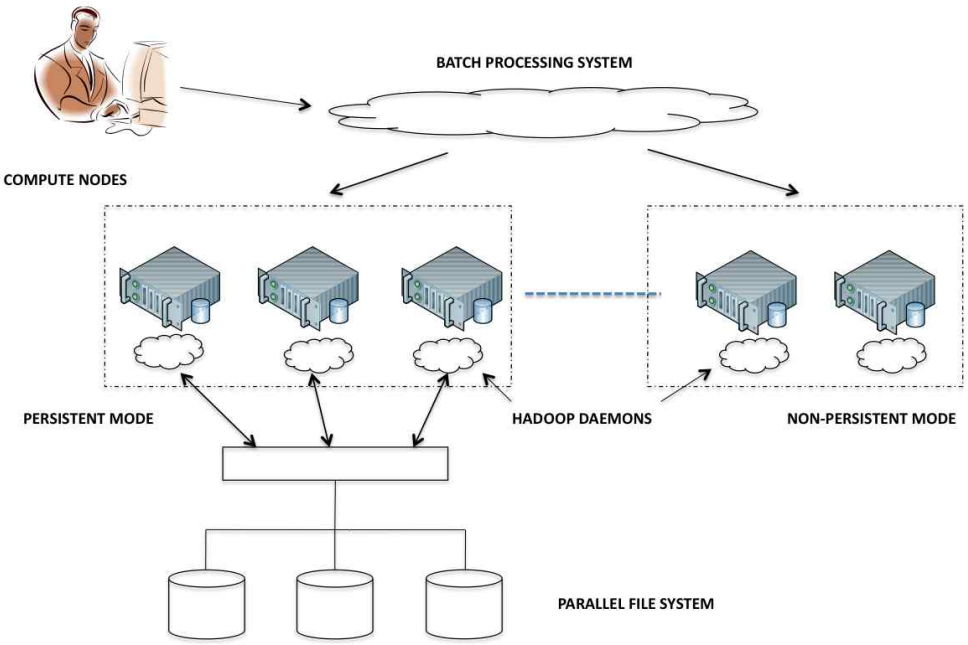

그림 8 myHadoop 아키텍처

myHadoop은 non-persistent 모드와 persistent 모드의 두가지 모드로 구성할 수 있다. non-persistent 모드에서는 하둡 데몬이 로컬 스토리지를 사용하여 HDFS를 구성하도록 한다. 이 모드에서는 두가지 문제가 있을 수 있는데 충분한 로컬 스토리지 용량이 존재하지 않을 수 있고, non-persistent 모드에서 실행된 하둡 작업의 결과는 하둡 작업이 종료되면 이용할 수 없게 될 수 있다. 이런 문제점을 극복하기 위해서 persistent 모드를 사용할 수 있는데 persistent 모드에서는 Lustre, GPFS와 같은 공유 파일시스템 상에서 HDFS를 구성할 수 있다.

다음 그림은 myHadoop의 실행 흐름을 나타낸 것이다. myHadoop은 별도의 클라이언트가 존재하지 않고 배치큐 스케줄러의 작업 스크립트 내부에서 실행되는 쉘스크립트인 myhadoop-configure.sh, myhadoop-cleanup.sh로 구현되어 있다.

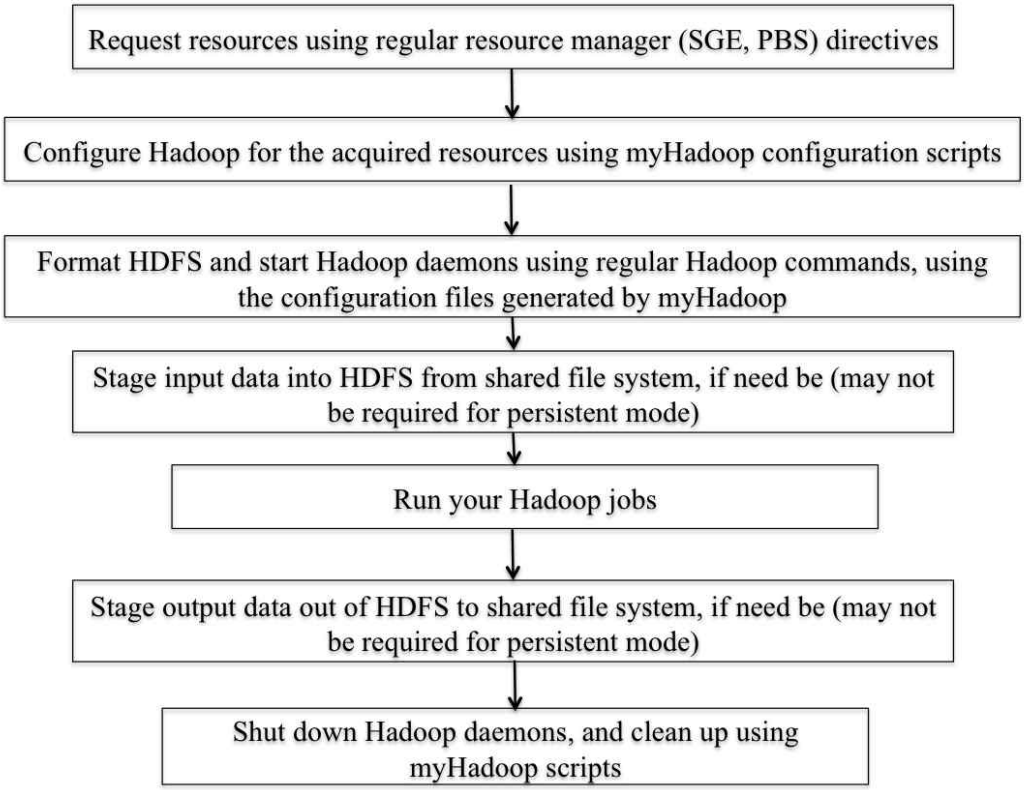

그림 9 myHadoop의 실행 흐름

myhadoop-configure.sh는 설정 디렉토리를 생성하고 배치큐 스케줄러로부터 할당받은 노드 정보를 바탕으로 하둡 설정 파일, 즉, masters, slaves, core-site.xml, hdfs-site.xml, mapred-site.xml을 생성하고 하둡 실행과 관련된 시스템 설정을 추가한다. myhadoop-cleanup.sh는 하둡 작업이 끝난 후에 할당받은 노드 상에 생성된 하둡 로그 파일을 사용자에게 반환하며 하둡 실행과 관련하여 변경된 시스템 설정을 해제한다. myhadoop-configure.sh와 myhadoop-cleanup.sh는 노드 간에 하둡 및 시스템 설정을 동기화하기 위해서 ssh를 사용한다.

제 3 절 myHadoop의 설치 및 테스트

여기서는 리눅스 기반으로 myHadoop을 설치하고 테스트하는 방법을 설명한다. myHadoop을 사용하기 위해서는 myHadoop과 하둡 배포판이 모든 노드에서 공유된 위치에 설치되어야 한다. 먼저 myHadoop 웹사이트로부터 myHadoop tarball를 다운로드받아서 압축을 해제한다. myHadoop 안에는 하둡 패치 파일이 존재하는데 이것은 myHadoop에서 하둡 설정 파일을 동적으로 구성하기 위해서 필요한 수정 사항을 포함하고 있다. 다음은 하둡과 myHadoop을 설치하고 하둡 패치파일을 적용하기 위한 과정을 보인 것이다. 하둡과 myHadoop은 ~/hadoop-stack에 설치되는 것으로 가정한다.

```
$ mkdir ~/hadoop-stack
$ cd ~/hadoop-stack
$ tar xvzf hadoop-1.2.1-bin.tar.gz
$ tar xvzf myhadoop-0.30b.tar.gz
$ cd hadoop-1.2.1/conf
$ patch < ../../myhadoop-0.30b/myhadoop-1.2.1.patch
```

한 예로 하둡 패치 파일 적용 후에 core-site.xml을 보면 다음과 같다.

```
$ cat core-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
  <name>hadoop.tmp.dir</name>
  <value>HADOOP_TMP_DIR</value>
  <description>A base for other temporary directories.</description>
</property>
<property>
  <name>fs.default.name</name>
```

<value>hdfs://MASTER_NODE:54310</value>

<description>The name of the default file system. A URI whose scheme and authority determine the FileSystem implementation. The uri's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority is used to determine the host, port, etc. for a filesystem.</description>

</property>

</configuration>

여기서 HADOOP_TMP_DIR, hdfs://MASTER_NODE:54310 등의 변경 사항을 확인할 수 있는데 이것은 myhadoop-configure.sh 내에서 동적으로 수정되는 부분들이다.

다음은 myhadoop-configure.sh의 persistent 모드 설정일 때의 동작과 관련된 부분을 보인 것이다.

```
if [ "z$PERSIST_BASE_DIR" != "z" ]; then
    ### Link HDFS data directories if persistent mode
    i=0
    for node in $(cat $HADOOP_CONF_DIR/slaves $HADOOP_CONF_DIR/masters | sort -u | head -n $NODES)
    do
        mkdir -p $PERSIST_BASE_DIR/$i
        echo "Linking $PERSIST_BASE_DIR/$i to ${config_subs[DFS_DATA_DIR]} on $node"
        ssh $node "mkdir -p $(dirname ${config_subs[DFS_DATA_DIR]}); ln -s $PERSIST_BASE_DIR/$i ${config_subs[DFS_DATA_DIR]}"
        let i++
    done

    ### Also link namenode data directory so we don't lose metadata on shutdown
```

```
        namedir=$(basename ${config_subs[DFS_NAME_DIR]})
        mkdir -p $PERSIST_BASE_DIR/$namedir
        for node in $(cat $HADOOP_CONF_DIR/masters | sort -u )
        do
            ssh $node "mkdir -p $(dirname ${config_subs[DFS_NAME_DIR]}); ln -s $PERSIST_BASE_DIR/$namedir ${config_subs[DFS_NAME_DIR]}"
        done
    fi
```

여기서 보면 persistent 모드인 경우 공유 파일시스템 상에 노드별 디렉토리를 생성하고 각 노드에서는 심볼릭 링크를 생성하여 모든 노드 상의 동일한 위치를 통해서 공유 파일 시스템 상의 노드별 디렉토리를 접근할 수 있게 한다.

이제 배치큐스케줄러의 작업스트립트를 생성하여 테스트할 차례이다. 다음은 배치큐스케줄러로 Torque가 사용된다고 가정하고 Torque의 작업 스크립트인 torque.qsub를 보인 것이다.

```
$ cat torque.qsub
#!/bin/bash
#################################################################################
#  torque.qsub - A sample submit script for Torque that illustrates how to
#     spin up a Hadoop cluster for a map/reduce task using myHadoop
#
#  Glenn K. Lockwood, San Diego Supercomputer Center      February 2014
#################################################################################
#PBS -q batch
#PBS -l nodes=3:ppn=1
#PBS -l walltime=1:00:00

cd $PBS_O_WORKDIR

### If these aren't already in your environment (e.g., .bashrc), you must define
```

```
### them.  We assume hadoop and myHadoop were installed in $HOME/hadoop-stack
export HADOOP_HOME=$HOME/hadoop-stack/hadoop-1.2.1
export PATH=$HADOOP_HOME/bin:$HOME/hadoop-stack/myhadoop-0.30b/bin:$PATH
export JAVA_HOME=/opt/jdk1.6.0_43
export HADOOP_CONF_DIR=$PBS_O_WORKDIR/hadoop-conf.$PBS_JOBID
```

myhadoop-configure.sh -c $HADOOP_CONF_DIR -s /tmp/$USER/$PBS_JOBID

```
if [ ! -f ./pg2701.txt ]; then
  echo "*** Retrieving some sample input data"
  wget 'http://www.gutenberg.org/cache/epub/2701/pg2701.txt'
fi

$HADOOP_HOME/bin/start-all.sh

$HADOOP_HOME/bin/hadoop dfs -mkdir data
$HADOOP_HOME/bin/hadoop dfs -put ./pg2701.txt data/
$HADOOP_HOME/bin/hadoop dfs -ls data
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-examples-*.jar wordcount data wordcount-output
$HADOOP_HOME/bin/hadoop dfs -ls wordcount-output
$HADOOP_HOME/bin/hadoop dfs -get wordcount-output ./

$HADOOP_HOME/bin/stop-all.sh
```

myhadoop-cleanup.sh

torque.qsub를 보면 스크립트 내에서 myhadoop-configure.sh와 myhadoop-cleanup.sh가 호출하는 것을 확인할 수 있다. myhadoop-configure.sh가 호출되기 전에 하둡과 관련된 환경변수들이 설정되어야 하며 myhadoop-configure.sh가 호출된 후에는 일반적인 하둡 명령

어를 사용하여 하둡 작업을 수행할 수 있다. 여기서는 wordcount 예제를 실행하는 가정하였다.

torque.qsub를 Torque에 제출하기 위해서는 Torque의 작업제출 명령어인 qsub를 이용하면 된다.

$ qsub torque.qsub

28.er001.ksc.re.kr

myHadoop을 통해서 wordcount 예제가 실행되는 과정은 torque.qsub시에 생성되는 torque.qsub.o[job id], torque.qsub.e[job id]을 통해서 확인할 수 있다.

제 4 절 SDSC의 myHadoop 활용 사례

myHadoop은 샌디에고 슈퍼컴퓨팅센터의 데이터 인텐시브 슈퍼컴퓨터인 Gordon에서 서비스되고 있다. 다음 그림은 Gordon에서 myHadoop을 통한 하둡 작업을 실행하는 방법을 설명한 웹사이트를 보인 것이다.

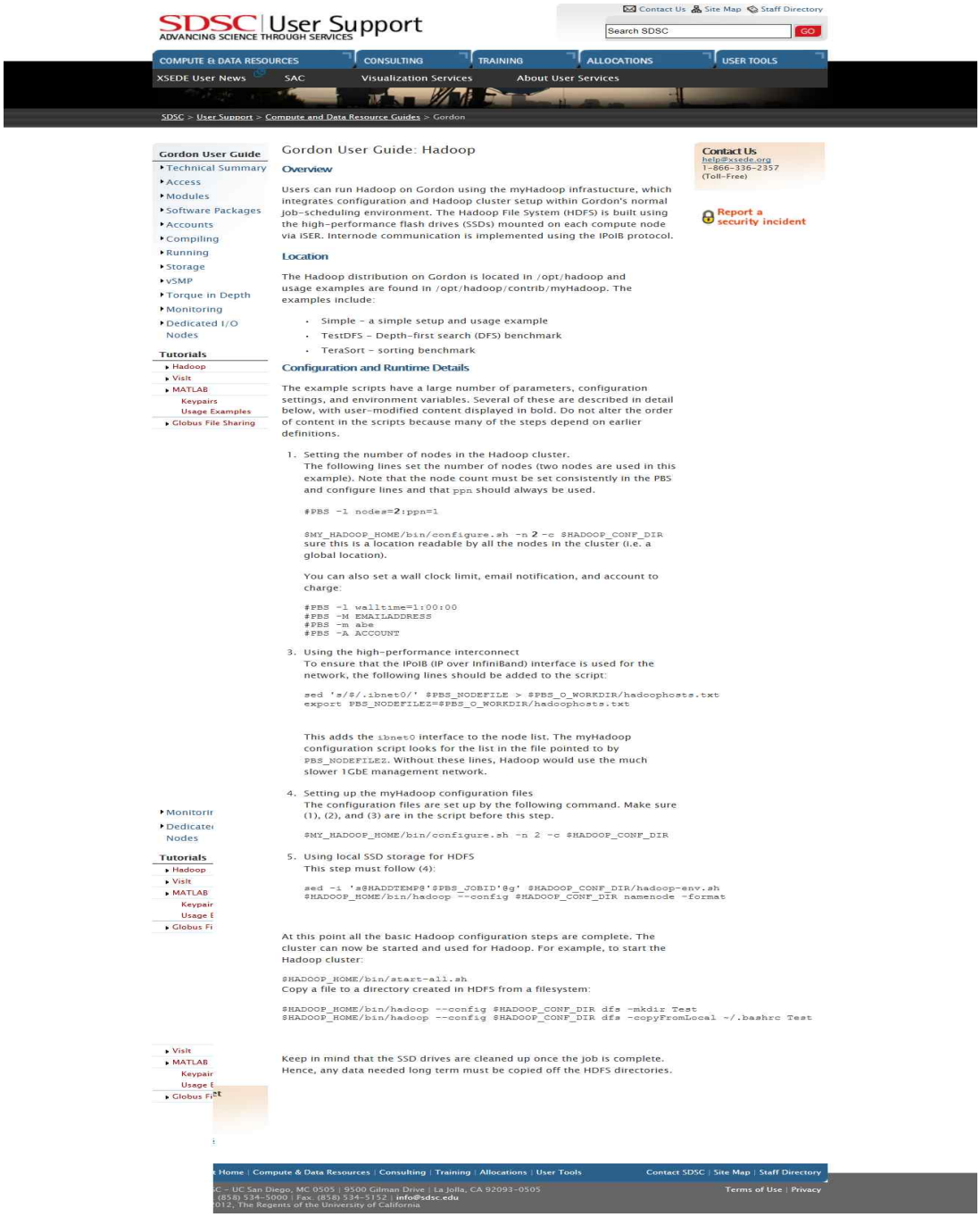

그림 10 Gordon의 하둡 서비스 관련 사이트

제 5 장 SpotHadoop

제 1 절 SpotHadoop 소개

데이터센터에서 사용되는 비즈니스 어플리케이션 환경, 슈퍼컴퓨팅센터에서 사용되는 과학 어플리케이션 환경에서는 SAN, NAS, 병렬분산파일시스템과 같은 공유된 스토리지 아키텍처를 가지고 인피니밴드와 같은 고성능 인터컨넥트로 연결되어 계산 클러스터 사이에서 공유된다. 계산 노드는 일반적으로 로컬 스토리지를 가지지 않으며 공유 파일시스템이 각 계산 노드에 마운트되어 사용되게 된다.

공유 스토리지 환경은 현재까지 많은 기관에서 이용되어 왔다. 이러한 환경에서는 계산 자원과 스토리지 자원을 프로비저닝하는데 있어서 더 많은 유연성을 가질 수 있고 일관된 스토리지 접근 시간을 가지고 데이터 지역성을 고려할 필요가 없으므로 어플리케이션을 단순화시키지만 계산 노드와 스토리지 노드 간의 QoS를 보장할 수 있는 충분한 대역폭을 요구하게 된다.

하둡은 Shared Nothing 구조를 기반으로 설계되었으며 1/10GbE 등의 범용 네트워크 기반으로 계산 노드의 로컬 디스크를 묶어서 분산 파일시스템을 구성하여 사용하고 있기 때문에 단순성, 저비용, 확장성과 관련하여 장점을 가지지만 어플리케이션 성능 관점에서 데이터 지역성이 주의 깊게 고려되어야 한다.

SpotHadoop은 오크리지 국립연구소에서 공유된 러스터 파일시스템 상에서 대규모 MPI 작업을 실행하는 클러스터 상에서 하둡 작업을 수행할 수 있는 연구 프로젝트로서 개발되었다. 이를 통해서 사용자는 데이터 분선 파이프라인 상에서 데이터를 공유하여 사용하는 것이 가능하다. SpotHadoop은 https://github.com/jhorey/SpotHadoop에서 다운로드 가능하다.

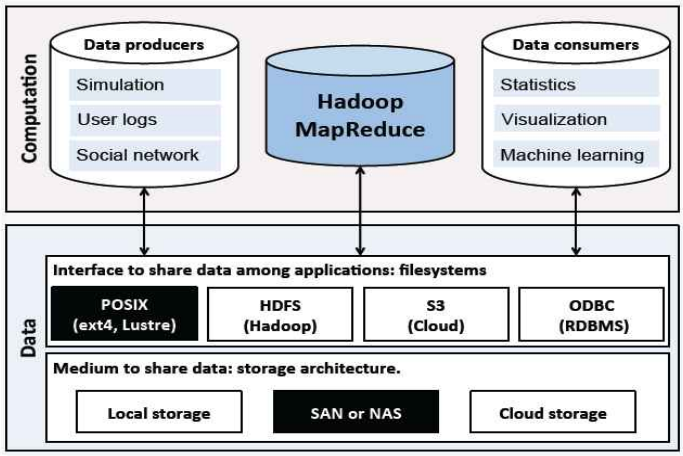

그림 11 데이터 분석 파이프라인을 위한 시스템 구성

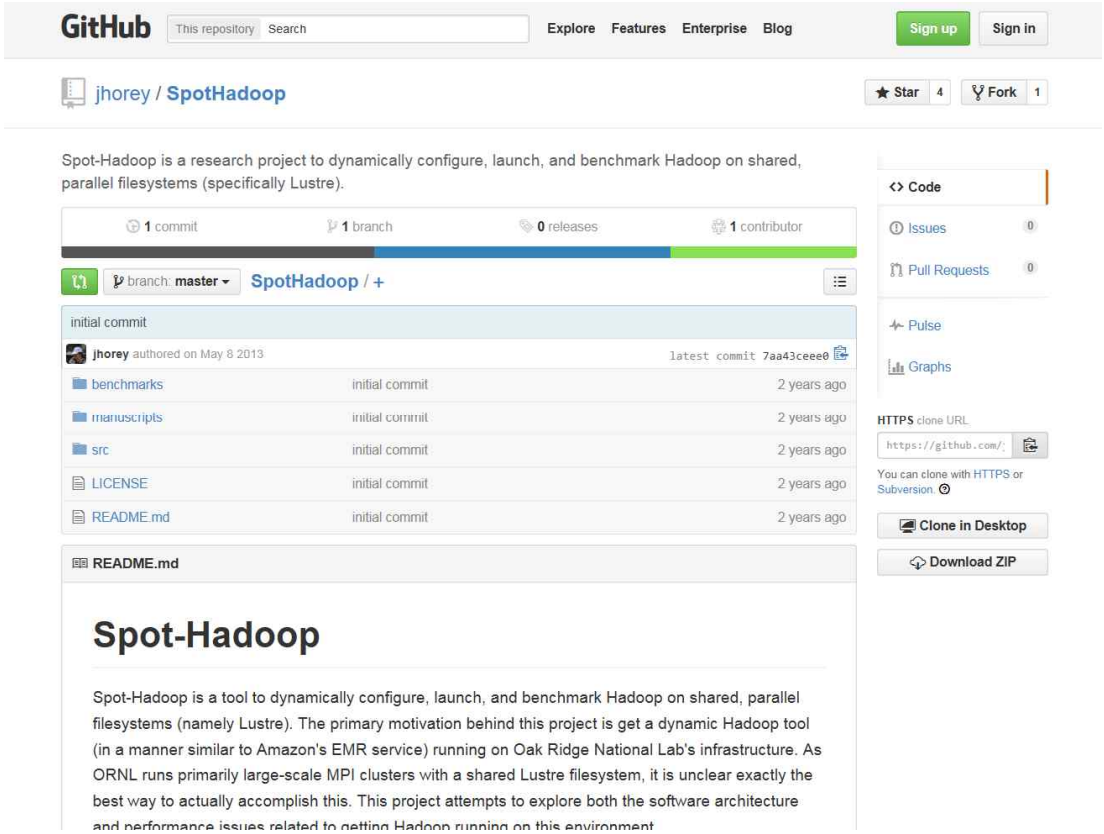

그림 12 SpotHadoop 웹사이트

제 2 절 SpotHadoop의 내부 구조

SpotHadoop은 배치큐스케줄러로 PBS(torque)만을 지원하며 plaunch은 open-mpi상에서만 실행된다. SpotHadoop은 오크리지 국립연구소의 슈퍼컴퓨터 자원인 smoky와 titan에서만 실행되도록 하드코딩된 부분이 존재하므로 타 슈퍼컴퓨터 자원에서 실행되기 위해서는 관련 부분들을 많이 수정할 필요가 있다.

SpotHadoop은 spot-hadoop, plaunch, hadooprun.py의 3가지 파일로 구성된다. spot-hadoop은 쉘스크립트 기반의 사용자 클라이언트로서 클러스터 상에 하둡을 위한 노드를 할당하고 하둡 작업을 실행하는 역할을 수행한다. plaunch는 mpi 기반의 헬퍼프로그램으로 클러스터 상에서 하둡이 셋업될 계산 노드의 호스트명, IP주소 등을 얻어오는 역할을 수행한다. hadooprun.py는 plaunch에서 전달된 네트워크 정보를 기반으로 환경 변수와 노드별 하둡 설정 파일을 생성하고 하둡 데몬을 실행하는 역할을 수행한다.

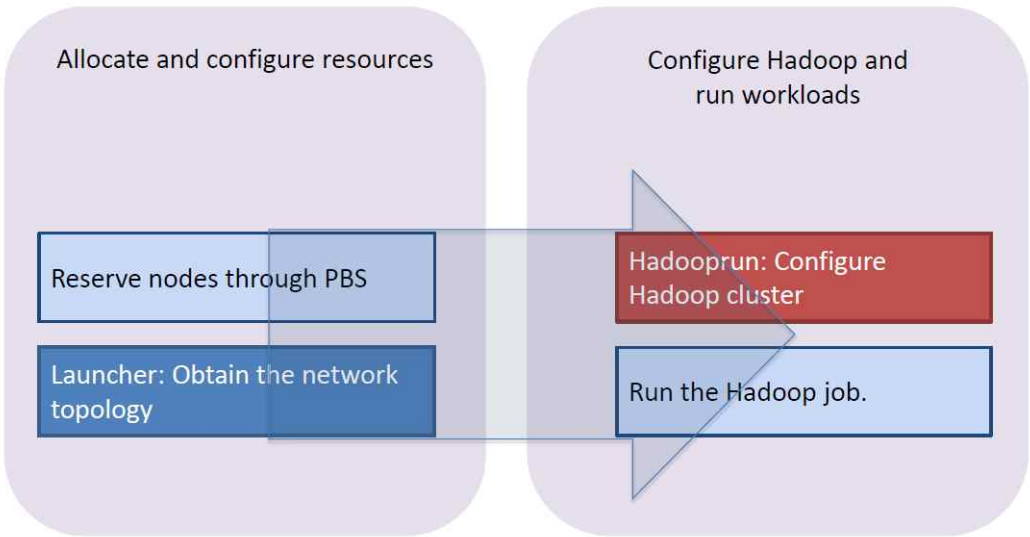

그림 13 SportHadoop의 PBS를 통한 하둡 동적 실행 구조

spot-hadoop은 다음과 같이 reserve, environment, hadoop, benchmark의 네가지 단계로 호출된다. SpotHadoop을 이용하여 하둡을 동적으로 실행하기 위해서는 spot-hadoop의 각 단계가 순차적으로 실행되어야 한다. reserve는 배치큐스케줄러를 이용하여 하둡이 실행될 노드를 할당하는 단계이다. reserve의 결과로 큐잉 시스템 상에 노드가 할당되고 노드 정보는 mpi 프로그램인 plaunch에 의해서 사용될 machinefile에 기록된다. environment는 하둡이 실행될 호스트의 네트워크 정보를 수집하는 단계이다. 호스트명 별로 파일이 생성되고 파일에는 물리 인터페이스별로 IP 주소가 기록된다. hadoop은 실제로 하둡이 할당된 노드 상에서 실행하기 위한 단계이다. 호스트명 별로 디렉토리가 생성되고 각 디렉토리에는 하둡

실행 파일이 복사되고 하둡 데몬이 environment 단계에서 생성된 노드 정보를 기반으로 각 노드에서 실행된다. benchmark는 생성된 하둡 상에서 벤치마크를 수행하는 단계로 exp.sh 라는 별도의 스크립트를 실행한다.

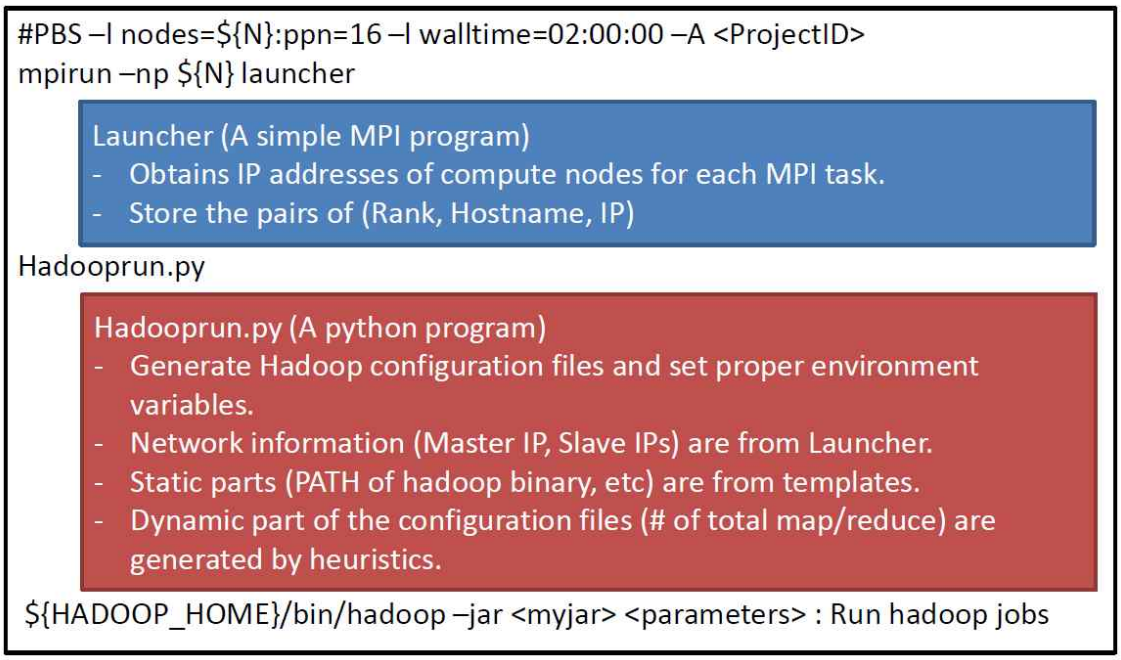

그림 14 SpotHadoop의 구조

제 3 절 SpotHadoop 설치 및 테스트

여기서는 리눅스 기반으로 SpotHadoop을 설치하고 테스트하는 방법을 설치한다. 앞서 기술한 것처럼 SpotHadoop은 오크리지 국립연구소의 슈퍼컴퓨터 자원을 대상으로 하둡을 동적으로 실행하기 위한 연구프로젝트로 진행된 소프트웨어이기 때문에 타 슈퍼컴퓨터 자원에서 실행하기 위해서는 소스 코드 수준에서 많은 변경이 필요하다.

SpotHadoop 웹사이트로부터 SpotHadoop tarball을 다운로드받아서 압축을 해제한다. SpotHadoop은 mpi프로그램인 plaunch를 포함하고 있고 이를 컴파일하기 위해서는 mpi라이브러리가 설치되어 있어야 하는데 여기서는 open-mpi가 설치되어 있는 것으로 가정한다. 다음과 같은 단계를 통해서 SpotHadoop을 설치한다.

```
$ unzip SpotHadoop-master.zip
$ cd SpotHadoop-master/src
$ make
```

먼저 SpotHadoop을 통해서 하둡이 실행될 노드를 할당하기 위해서 spot-hadoop의 reserve옵션을 사용하여 다음과 같이 실행한다.

```
$ ./spot-hadoop reserve batch
Python 2.6.6
111.er001.ksc.re.kr
```

PBS(torque)의 qstat을 통해서 할당된 노드 정보를 확인할 수 있는데 여기서는 er002, er003, er004의 노드가 할당되어 있음을 확인할 수 있다.

```
$ qstat -an

er001.ksc.re.kr:

Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname         SessID  NDS
TSK   Memory   Time    S   Time
-----------------------  ----------  --------  ----------------  ------
------ ------ ------ --------- - ---------
```

111.er001.ksc.re.kr ltest batch pbs.script 0 3 3 -- 01:00:00 R 00:00:02

er002+er004+er003

다음은 할당된 노드의 네트워크 정보를 수집하기 위해서 spot-hadoop의 environment옵션을 사용하여 다음과 같이 실행한다.

```
$ ./spot-hadoop environment
Python 2.6.6
removed `/home/ltest/data/network/er002.ksc.re.kr'
removed `/home/ltest/data/network/er003.ksc.re.kr'
removed `/home/ltest/data/network/er004.ksc.re.kr'
removed `/home/ltest/data/network/hosts'
attempting local write /home/ltest/data/network/er002.ksc.re.kr
attempting local write /home/ltest/data/network/er003.ksc.re.kr
attempting local write /home/ltest/data/network/er004.ksc.re.kr
attempting write /home/ltest/data/network/hosts
```

network디렉토리의 호스트명 별로 파일이 생성되고 각 파일에는 호스트의 네트워크 정보가 저장되어 있음을 확인할 수 있다. 다음은 er002.ksc.re.kr 파일을 예로 보인 것이다.

```
$ cat er002.ksc.re.kr
lo:127.0.0.1
eth1:192.168.0.102
ib0:10.0.0.102
mpi:0
```

다음은 하둡을 실행하기 위해서 spot-hadoop의 hadoop옵션을 사용하여 다음과 같이 실행한다.

```
$ ./spot-hadoop hadoop
Python 2.6.6
```

clear metadata for previous cluster setup

setup network environment.

setup hadoop configuration and start daemons.

er004.ksc.re.kr cmd: python /home/ltest/bin/hadooprun.py -e /home/ltest/data/environment > er004.ksc.re.kr.out 2> er004.ksc.re.kr.err

er002.ksc.re.kr cmd: python /home/ltest/bin/hadooprun.py -e /home/ltest/data/environment > er002.ksc.re.kr.out 2> er002.ksc.re.kr.err

er003.ksc.re.kr cmd: python /home/ltest/bin/hadooprun.py -e /home/ltest/data/environment > er003.ksc.re.kr.out 2> er003.ksc.re.kr.err

하둡의 실행 상태는 생성된 호스트명.out, 호스트명.err을 통해서 확인할 수 있다. 다음은 er002.ksc.re.kr.out파일을 예로 보인 것이다.

$ cat er002.ksc.re.kr.out

starting jobtracker, logging to /home/ltest/hadoop/hadoop-1.2.1/libexec/../logs/hadoop-ltest-jobtracker-er001.ksc.re.kr.out

JOBTRACKER er002.ksc.re.kr

create_masters: er002.ksc.re.kr

could not get ip address for 0

hadoop-daemon.sh --config /home/ltest/hadoop/conf/er002.ksc.re.kr start jobtracker

하둡과 관련된 명령어를 hadoop 명령어에서 --config옵션을 통해 하둡 설정 파일의 위치를 지정하여 사용하면 된다. 여기서는 하둡을 이용하여 디렉토리를 리스팅하는 예를 보인 것이다.

$ bin/hadoop --config /home/ltest/hadoop/conf/er002.ksc.re.kr/ fs -ls

Found 26 items

drwxr-xr-x - ltest ltest 4096 2014-11-16 14:09 /home/ltest/hadoop/hadoop-1.2.1/bin

drwxr-xr-x - ltest ltest 4096 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/webapps

drwxr-xr-x - ltest ltest 4096 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/share

-rw-rw-r— 1 ltest ltest 10525 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/ivy.xml

drwxr-xr-x - ltest ltest 4096 2014-11-15 10:13 /home/ltest/hadoop/hadoop-1.2.1/sbin

```
-rw-rw-r--  1 ltest ltest   498744 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/CHANGES.txt
drwxrwxr-x  - ltest ltest     4096 2014-11-16 11:58 /home/ltest/hadoop/hadoop-1.2.1/logs
-rw-rw-r--  1 ltest ltest      414 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/hadoop-client-1.2.1.jar
-rw-rw-r--  1 ltest ltest     6842 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/hadoop-ant-1.2.1.jar
-rw-rw-r--  1 ltest ltest    13366 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/LICENSE.txt
drwxr-xr-x  - ltest ltest     4096 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/contrib
drwxr-xr-x  - ltest ltest     4096 2014-11-15 10:13 /home/ltest/hadoop/hadoop-1.2.1/lib
-rw-rw-r--  1 ltest ltest   385634 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/hadoop-tools-1.2.1.jar
drwxr-xr-x  - ltest ltest     4096 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/c++
-rw-rw-r--  1 ltest ltest  4203147 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/hadoop-core-1.2.1.jar
-rw-rw-r--  1 ltest ltest  3126576 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/hadoop-test-1.2.1.jar
drwxrwxr-x  - ltest ltest     4096 2014-11-15 10:13 /home/ltest/hadoop/hadoop-1.2.1/src
drwxr-xr-x  - ltest ltest     4096 2014-11-15 10:13 /home/ltest/hadoop/hadoop-1.2.1/ivy
-rw-rw-r--  1 ltest ltest   142726 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/hadoop-examples-1.2.1.jar
-rw-rw-r--  1 ltest ltest      101 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/NOTICE.txt
drwxr-xr-x  - ltest ltest     4096 2014-11-15 10:13 /home/ltest/hadoop/hadoop-1.2.1/libexec
drwxr-xr-x  - ltest ltest     4096 2014-11-15 10:13 /home/ltest/hadoop/hadoop-1.2.1/.eclipse.templates
-rw-rw-r--  1 ltest ltest     1366 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/README.txt
-rw-rw-r--  1 ltest ltest      417 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/hadoop-minicluster-1.2.1.jar
drwxr-xr-x  - ltest ltest     4096 2014-11-16 13:09 /home/ltest/hadoop/hadoop-1.2.1/conf
-rw-rw-r--  1 ltest ltest   121130 2013-07-23 07:26 /home/ltest/hadoop/hadoop-1.2.1/build.xml
```

제 4 절 ORNL의 SpotHadoop 활용 사례

SpotHadoop은 오크리지 국립 연구소의 슈퍼컴퓨팅자원을 대상으로 하둡을 동적으로 실행하기 위한 연구 프로젝트로서 진행되고 있다. 오크리지 국립 연구소의 슈퍼컴퓨팅자원은 스파이더(Spider)라는 러스터 병렬분산파일시스템을 기반으로 하고 있으며 SpoHadoop은 스파이더 파일시스템 기반의 슈퍼컴퓨팅 자원에서 실행될 수 있다.

그림 15 오크리지 국립연구소의 스파이더 파일시스템

제 6 장 HOD, myHadoop, SpotHadoop 기술 비교

다음 표는 지금까지 분석한 해외 슈퍼컴퓨팅센터의 3가지 동적 하둡 실행 기술인 HOD, myHadoop, SpotHadoop을 비교 정리한 것이다.

표 1 HOD, myHadoop, SpotHadoop 기술 비교

	myHadoop	HOD	SpotHadoop
Development	SDSC	Apache	ORNL
Batch Queue System Support	Torque, SGE, Slurm	Torque	PBS
Language	Shell script	Python	Shell script, Python, C
Components	myhadoop-configure.sh myhadoop-cleanup.sh	hod (client) RingMaster (manager) HODRing (agent)	hadooprun.py (hadoop executor) plaunch (parallel launcher) spot-hadoop (client)
User interface	No (called in job script)	CLI	CLI
Shared Filesystem Support	Yes	Yes (external HDFS only)	No
Config Management	No	Centralized (hodrc)	No
Log Management	Per-user	Centralized	No
Lifetime management	Yes	Yes	No
Case of Application	SDSC	University of Chicago	ORNL

제 7 장 결론

본 보고서에서는 배치큐스케줄러에 의해서 관리되는 클러스터 환경에서 하둡을 동적으로 프로비저닝하여 사용할 수 있는 몇가지 기술들에 대해서 분석하였다. 본 보고서에서는 아파치 커뮤니티 등의 글로벌 오픈소스 커뮤니티와 미국 슈퍼컴퓨팅센터의 사례를 바탕으로 세 가지 기술, HOD(Apache), myHadoop(SDSC), SpotHadoop(ORNL)을 선정하였으며 각각의 기술을 분석하고 비교하였고, 기술 소개, 내부 구조, 설치 및 테스트, 활용 사례 등을 기술하였다.

하둡은 현재 과학 응용 분야 보다는 기업 응용 분야의 데이터 집약형 응용 연구를 위한 프레임워크로서 널리 사용되고 있다. 시스템 아키텍처 상의 차이로 인해 하둡을 과학 응용이 실행되는 배치큐스케줄러 기반의 고성능컴퓨팅 환경에서 직접적으로 사용하는 것은 어렵지만 현재 다양한 툴들이 나와 있고 적용 및 서비스 가능성에 대해서 다양한 실험들이 이루어지고 있는 것으로 파악된다. 이런 실험의 결과로서 적용 및 서비스 가능성이 검증된다면 고성능컴퓨팅 환경에서 시뮬레이션 위주의 계산 집약적인 응용과 함께 하둡과 같은 오픈소스 분산데이터 처리 프레임워크를 기반으로 데이터 집약적인 응용까지 서비스할 수 있는 토대가 될 것으로 예상된다.

참 고 문 헌

[1] Apache Hadoop, http://hadoop.apache.org/

[2] Apache HOD, http://hadoop.apache.org/docs/r1.2.1/hod_scheduler.html

[3] myHadoop, https://github.com/glennklockwood/myhadoop/

[4] SpotHadoop, https://github.com/jhorey/SpotHadoop

[5] Torque, http://www.adaptivecomputing.com/products/open-source/torque/

[6] Slurm, http://slurm.schedmd.com/

[7] Lustre, http://opensfs.org/

[8] Tom White, Hadoop: The Definitive Guide, O'REILLY, 2010.

[9] Top500 Supercomputing System, http://www.top500.org