K-ONE 기술 문서 #3

OpenStack과 Apache Mesos를 활용한 클러스터 스케줄링과 활용사례

Document No. K-ONE #3
Version 0.2
Date 2016. 05. 08.

Author(s) 김남곤

K-ONE

ONK
OpenNetworkingKorea

■ 문서의 연혁

버전	날짜	작성 내용	비고
0.1	2016. 04. 01.	문서구성 및 아웃라인	
0.2	2016. 05. 08.	초안 작성 (일부 보완필요)	

본 문서는 2015년도 정부(미래창조과학부)의 재원으로 정보통신 기술진흥센터의 지원을 받아 수행된 연구임 (No. B0190-15-2012, 글로벌 SDN/NFV 공개소프트웨어 핵심 모듈/기능 개발)

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. B0190-15-2012, Global SDN/NFV OpenSource Software Core Module/Function Development)

기술문서 요약

BigData/HPC를 위한 하드웨어 등 인프라 기술이 발전하고 다양한 응용 서비스와 요구사항을 수용하기 위한 클러스터링 기술은 특정 응용이나 서비스를 위해 설정된 Dedicated Cluster 환경은 점차 Cloud Computing 기반의 Shared Cluster 체계로 변하고 있다. 본 기술문서는 이러한 기술개발 방향에서 자원의 클라우드 인프라를 관리하는 OpenStack 측면에서는 Bare-metal Compute Service인 Ironic을 활용하고, 프로비전된 클러스터의 자원 관리자로 Apache Mesos를 설치하고 설정 후 Apache Spark를 운용하는 기술적 원리와 경험을 다룬다. 다시 말해, Mesos Cluster를 구성하는 Mesos Master와 Slave 구조에서 Master에 내장된 자원 할당자의 동작 방법과 정책, Mesos가 제공하는 자원들을 활용하여 각각의 Framework가 활용하는 일련의 과정들의 이해를 도울 수 있도록 작성하였다. 또한 자동화된 Mesos 설치와 설정프로그램으로 구성된 Mesos Cluster에서 Data Analytic engine 중 하나인 Spark를 운용하는 예제로 Spark-Perf를 활용한 벤치마킹 사례를 제시하였다.

Contents

K-ONE #3. OpenStack과 Apache Mesos를 활용한 클러스터 리소스 스케줄링과 활용사례

그림 목차

표 목차

K-ONE #3. OpenStack과 Apache Mesos를 활용한 클러스터 리소스 스케줄링과 활용사례

1. BigData/HPC를 위한 클러스터링 개요

o 많은 양의 데이터 또는 고성능의 연산을 위해서 다수의 컴퓨터를 연동하여 처리하는 클러스터링은 일반화되어 있는 방법이다. BigData 처리는 Hadoop 기반 Map-Reduce를 시작으로 BigData 처리가 대중화되었다. BigData 처리를 위한 특별한 고성능 머신이 아니라도 일반적인 x86 플랫폼에서 다수의 머신을 클러스터링하는 방법론을 적용하였다. Hadoop 1.x 버전에서는 단일 클러스터에서 한번에 하나의 Job만을 실행시킬 수 있었다. 그러나 Testbed와 Production의 공존필요성, 실시간성 데이터에 대한 처리의 필요성 등이 요구되면서 클러스터를 공유할 수 있는 여러 가지 방법들이 제안되었다.

o 한편 자연과학 등의 병렬연산이 많이 사용되는 분야에서는 병렬 프로그래밍 라이브러리인 Message Passing Interface (MPI)를 기반으로 여러 대의 머신을 클러스터링하여 주어진 연산 또는 문제를 빠르게 해결하는 방법이 일반적이며, 최대한 빠른 실행결과를 얻기위한 High Performance Computing (HPC) 영역으로 볼 수 있다.

o BigData/HPC를 위한 클러스터를 구축하기 위해서는 공통적으로 요구되는 관리기능은 자원 (Resource)과 작업 (Job)관리 기능이다. 하지만 BigData와 HPC 작업의 관리기능의 요구사항은 다르다. HPC는 특정한 하나의 Job을 최대한 빠르게 처리하는 (dedicated) 특징이 있다. 한편 BigData 처리의 경우 연산성능이 빠를수록 좋지만, 일반적으로 결과를 실시간으로 요구하지도 않으며 다른 작업과 동시에 실행할 경우 우선순위도 낮게 부여하여 처리의 “속도”보다는 “가능성”에 좀 더 가깝다고 볼 수 있다.

o x86 시스템을 중심으로 구성하는 Cloud Computing 환경에서 BigData 처리는 하나의 대중적인 서비스로 자리를 잡았고 Hortonworks, Databrix 등 Analytics Service를 비즈니스 모델로 영업하는 회사도 등장하였다. 아직 Cloud 기반의 HPC 서비스는 초기단계로 BigData 처리 서비스만큼 대중화 되지는 않았지만 현재의 기술적 흐름은 HPC-over-Cloud로 향하고 있다. 각 서비스 또는 사용자들의 Workload를 수용할 수 있는 자원들의 준비와 구성이 유연할 뿐만 아니라 실제 머신들을 다양한 목적으로 설정하여 효율적으로 사용할 수 있기 때문이다.

o 오픈소스 Cloud OS의 대표 격인 OpenStack 프로젝트[1]는 가상머신 (VM)을 기반으로 개발이 시작되었으나, 최근버전은 Compute (Worker)노드를 가상머신이 아닌 Bare-metal 머신으로 활용하는 Ironic 프로젝트가 추가 되었다. 한편 가상화 기술의 다른 흐름으로, VM보다 자원사용의 효율성을 증가시키고 실행성능이 좋

은 Linux Container도 OpenStack에 통합하려는 노력이 있다. 하지만 OpenStack 기반의 Linux Container 관련기술의 성숙도와 Legacy HPC Service/Workload 등을 고려하여 본 기술문서에서는 Bare-metal 머신 기반의 OpenStack을 적용하고자 한다.

o 클러스터 메니저 측면에서는 오픈소스 클러스터 관리프로그램인 Apache Mesos를 사용한다. BigData/HPC 처리를 위한 클러스터를 OpenStack Ironic 환경에서 Provisioning 하기 위해, 자동 설정되도록 구성하는 프로그램의 설계와 프로토타입, 그리고 동작원리와 활용사례를 기술하여 정리하고자 한다.

o 본 기술문서는 Playground (테스트베드)에서 구현한 프로토타입을 운영하면서 얻은 노하우와 관련 자료를 바탕으로 클러스터 리소스 스케줄링에 대한 기초적인 수준의 기술을 분석하여, 관련기술이 생소한 사람에게 기술적 이슈를 이해할 수 있는 참고자료로 사용할 수 있도록 한다. 또한 해당 분야에 관심있는 연구자들과 정보를 공유하며 협업 및 향후 연구개발을 위한 기초자료로 활용되기를 기대 한다.

2. OpenStack Ironic과 Mesos Provisioning

2.1. OpenStack Ironic 소개

o Ironic [2]은 가상화로 인한 성능저하를 최대한 방지하기 위하여 Bere-metal 리소스와 OpenStack의 인프라 (자원)관리를 통합하기 위해 개발하는 프로젝트이다. <그림 1>은 Ironic의 논리적인 구조를 나타낸 것으로, 기존의 Compute Service인 Nova를 확장하여 활용하고 각각의 Bere-metal 노드는 하나의 Driver로 취급한다. 다시 말해 가상머신 기반의 Nova-Compute 노드가 사용하는 하이퍼바이저 종류에 따라 Nova Compute Driver를 설정하는 것과 유사하다.

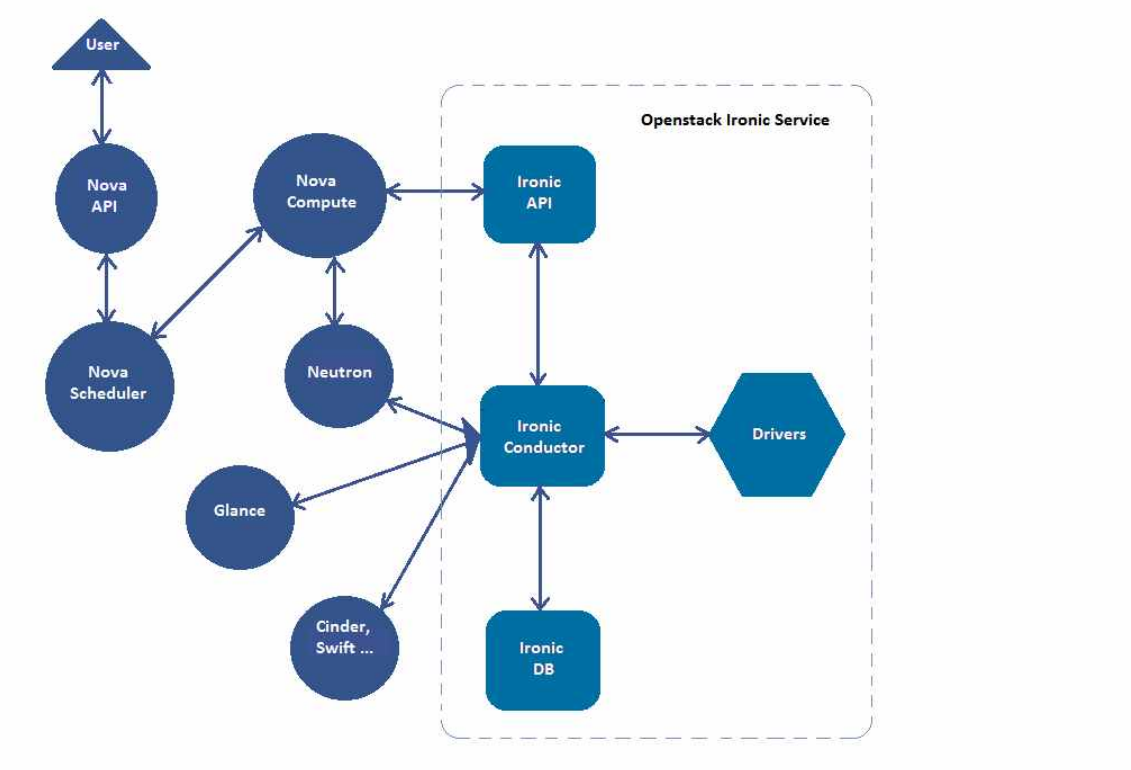

그림 1. OpenStack Ironic Logical Architecture [2]

o Ironic Driver의 종류는 pxe_ipmitool, pxe_ssh, agent_ipmitool, agent_ssh 등이 있으며, 기타 3rd party Driver도 사용할 수 있다. Drive를 중심으로 Ironic 노드를 생성하고 사용하고자하는 이미지를 등록하여 사용한다. Ironic의 이미지는 Deploy 이미지와 User 이미지가 있는데, Deploy 이미지는 어떠한 OS도 설치되지 않은 머신에 User 이미지를 배포하기 위한 커널과 램디스크 이미지로 구성된다. User 이미지는 각 사용자가 정의한 머신의 이미지로서 실제 물리자원들에 설치되는 운영체제이다.

o Ironic 또한 OpenStack의 이미지 서비스인 Glance를 활용하는데, Deploy 이미지

와 User 이미지 모두 Glans에 등록하고 리턴되는 UUID를 Node에 등록한다.

2.2. Playground for OpenStack Ironic and Mesos

o OpenStack Ironic과 Mesos를 실험하기 위한 Playground (SmartX Type-M Testbed)의 구성은 아래 <그림 2>와 같이 나타내었다. 크게 Playground를 총괄하고 관리하는 DevTower와 각 Worker로 나눌 수 있다. DevTower에는 OpenStack Controller와 Mesos Master가 실행된다.

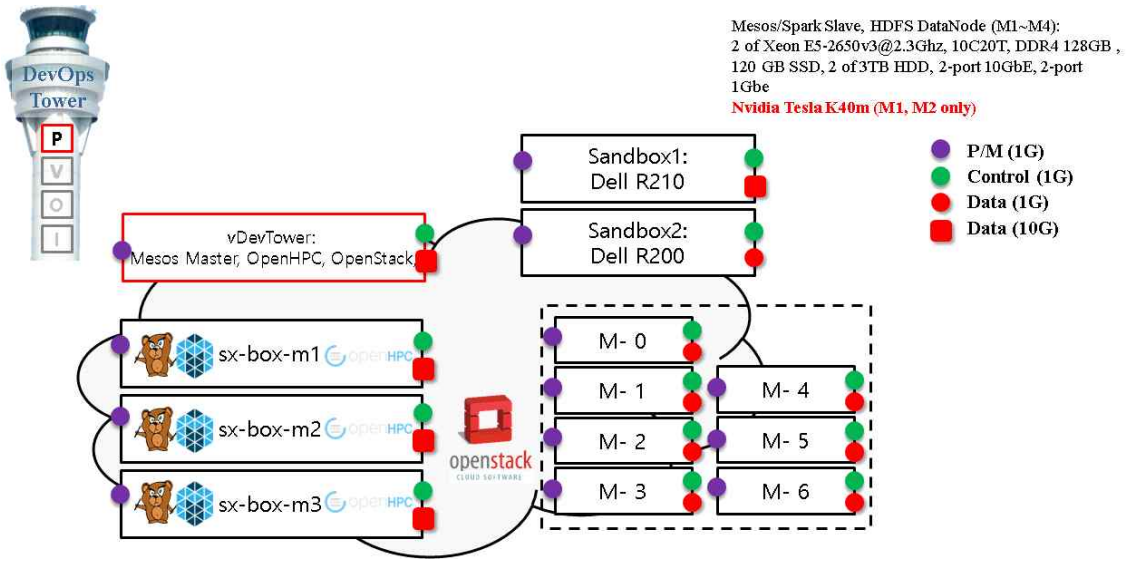

그림 2. SmartX Type-M Playground

o SmartX Type-M Playground를 구성하는 주요 Worker 노드의 하드웨어 사양은 아래 <표 1>과 같다.

표 1. Playground 주요 Worker 노드의 하드웨어 사양

구분	사양	비고
CPU	2 of Xeon E5-2650v3@2.3Ghz, 10C20T	
RAM	DDR4 128GB, RDIMM	
Storage	120 GB SSD, 2 of 3TB HDD	
Network	2-port 10GbE, 2-port 1Gbe	
GPU	Nvidia Tesla K40m	for 2 machines

o 대상 운영체제는 Ubuntu와 CentOS를 대상으로 한다. Ubuntu는 최신버전의 커널을 채택하고 많이 사용하는 패키지를 기본으로 채택하고, 패키지 관리가 쉬워서 사용이 편리하므로 개발용으로 많이 사용하는 편이다. 그리고 Ubuntu는 6개월 단위로 릴리즈 되지만 대부분의 경우 2년 단위로 릴리즈 되는 Long Term

Support (LTS) 버전을 권장하며 본 문서에서도 다른 언급이 없는 한 14.04 LTS 버전을 다룬다. CentOS는 RedHat Enterprise Linux (RHEL)의 오픈버전으로 몇 가지 RHEL의 특화된 기능을 제외하고는 동일하다. 기업용 버전을 대상으로 하는 만큼 버전별로 지원기간이 10년으로 길고 안정적이다. 또한 배포판 자체는 기본 패키지들을 최소화하여 각 시스템의 용도나 운영에 따라 커스터마이징하여 사용하도록 되어 있다.

2.3. Mesos Cluster Provisioning

o VM기반의 OpenStack에서 Mesos Cluster를 구성하거나, 특정 물리머신에 Dedicated Mesos Cluster를 구성할 수도 있다. 하지만 본 기술문서의 목적에 따라 OpenStack Ironic을 설정하고 운용하는 것에 대한 이슈를 정리한다. 또한 동적으로 Mesos Cluster를 구성하기 위한 자동화기법이 적용된 Mesos [3] 설치 및 설정 프로그램에 대해 기술한다.

o [확인/수정 필요] Mesos를 설치하고 구성하는 방법은 여러 가지가 있겠지만 OpenStack Ironic 이미지의 숫자를 줄여서 관리를 용이하게 하면서, 확보된 중앙제어 기능을 활용하여 User Image가 배포된 후에 Mesos를 설치하고 환경설정 한다. OS가 설치된 후에 박스의 역할(마스터 또는 슬레이브)에 맞는 설치 스크립트를 배포한다. <그림 3>은 이러한 Mesos 자동 설치 프로그램의 배포개념을 나타내고 있다. Mesos의 환경설정을 위해 해당 박스의 정보 (이름, IP 주소 등)를 알아야 하며, 해당 정보를 옵션으로 입력하여 설치 프로그램을 실행한다.

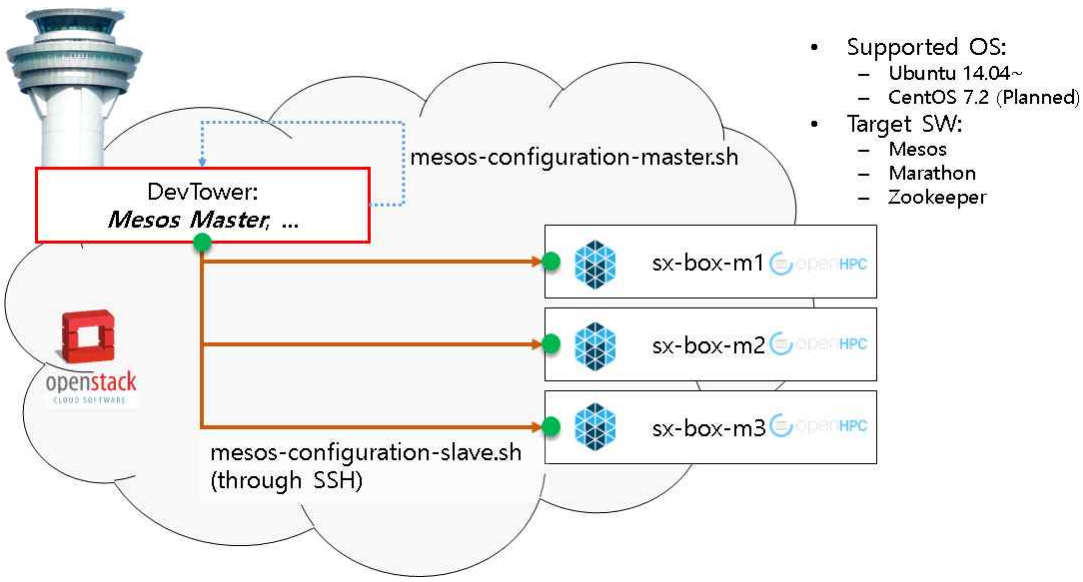

그림 3. 자동화된 Mesos 설치 및 설정 프로그램 배포개념

3. Mesos 활용: Mesos Framework

3.1. Apache Mesos 소개

o Mesos는 “하나의 클러스터에서 2개 이상의 Hadoop을 실행할 수 없을까?”라는 작은 아이디어에서 출발하였다. Google의 클러스터 메니저인 “Borg“ 그리고 멀티 코어 CPU와 운영체제 (Linux)에서 영감을 얻어 버클리의 학생들이 수업프로젝트로 제안한 ”Nexus“를 시작으로 기술문서, 논문발표, Apache incubating 순으로 진행되었다. Mesos는 여러대의 머신이 마치 하나의 머신인 것처럼 관리할 수 있는 데이터센터의 Kernel을 지향하고 있으며, 하나의 Mesos Cluster에서 여러개의 Hadoop과 같은 Framework 들이 동적으로 잘 실행될 수 있도록 하는 클러스터 메니저이다.

o 운영체제만 가지고는 응용 서비스가 실행되지 않는 것처럼 Mesos만 가지고는 실제 응용 서비스가 실행되지 않는다. Mesos 위에서 실행할 수 있는 Framework 들이 요구되는데, 다양한 형태와 목적을 가진 Framework 들이 오픈소스로 개발되고 있다. Hadoop 2.0부터 적용된 자원 관리자인 YARN (Yet Another Resource Negotiator)과 더불어 2-level scheduler 구조를 가지고 있다. 또한 자원의 할당 관점에서는 중앙집중형 (Centralized) 스케줄러이다. Mesos Cluster를 구성하는 요소는 Master(s)와 Slave(s)이며, Mesos-Master의 안정적인 동작과 Fault-tolerant를 위해 여분의 Master 인스턴스를 실행시켜고 ZooKeeper [4]를 활용하여 리더를 선택하는 방법을 사용한다. 2-level scheduler 관점에서 Mesos는 현재 가용한 자원들을 Master에 등록된 Framework 들에게 제공 (offer)하고, 사용을 희망하는 Framework은 자원할당을 요청한다 (1st level). 자원을 할당받은 각 Mesos Framework들은 실행하려는 Job 또는 Task에 대해 자체적으로 자원을 배분하고 작업들을 스케줄링한다 (2nd level).

o <그림 4>는 Mesos의 자원할당 관점에서 Mesos 마스터와 프레임워크들간의 관계를 나타낸 것이다. Mesos 마스터에 기본으로 자원할당모듈이 적용되어 있고, 여러 옵션들을 조합하여 스케줄링 정책을 만들 수 있다. 만약 사용자 정의 자원할당 모듈을 사용하려면, 정의된 API를 활용하여 구현하고 Master 실행시에 해당 모듈을 자원할당자로 지정한다.

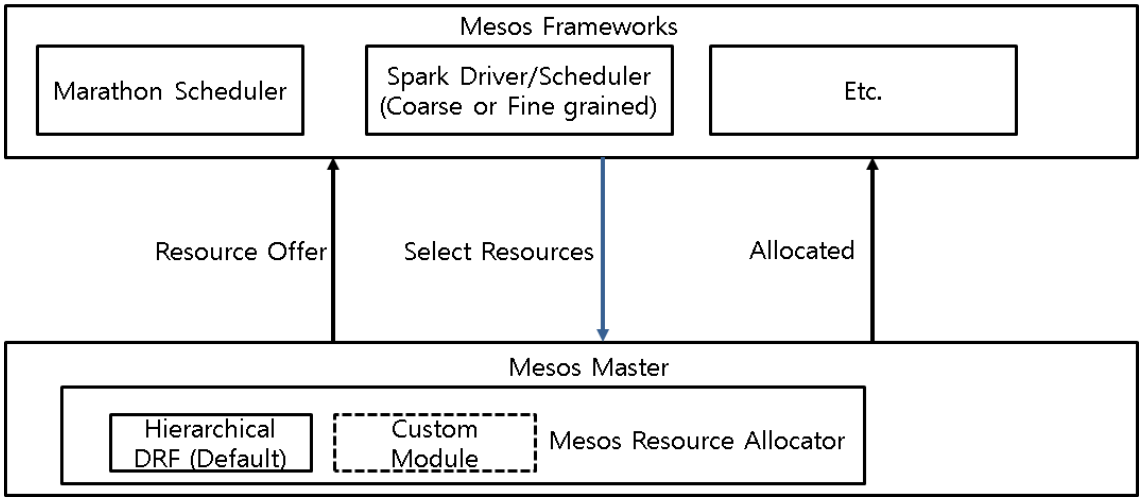

그림 4. Mesos 리소스 할당자와 2-level 자원 스케줄링

3.2. Apache Mesos Internals

o Mesos의 자원 스케줄링을 위한 기본모듈로 Hierachiracical Dorminant Resurce Fairness (HDRF) [4]를 사용하고 있고, 그림은 HDRF의 기본이 되는 DRF [5]를 나타낸 것이다. 어떤 사용자나 Framework마다 요구하는 자원의 종류와 양이 다르기 때문에 Shared Cluster에서 Framework별 공정한 자원할당을 위해 자원 중 가장 많이 사용되는 것 (Dominant Resource)를 기준으로 정한다. 만약 <그림 5>와 같이 사용자가 자원요구량을 변화시키면 다시 계산하여 반영하는 상황에서 CEEI는 다른 자원의 할당량에도 영향을 미치지만 DRF는 그렇지 않아서 훨씬 효율적이다. 그리고 <그림 6>은 DRF의 알고리즘을 기술한 슈도코드이다.

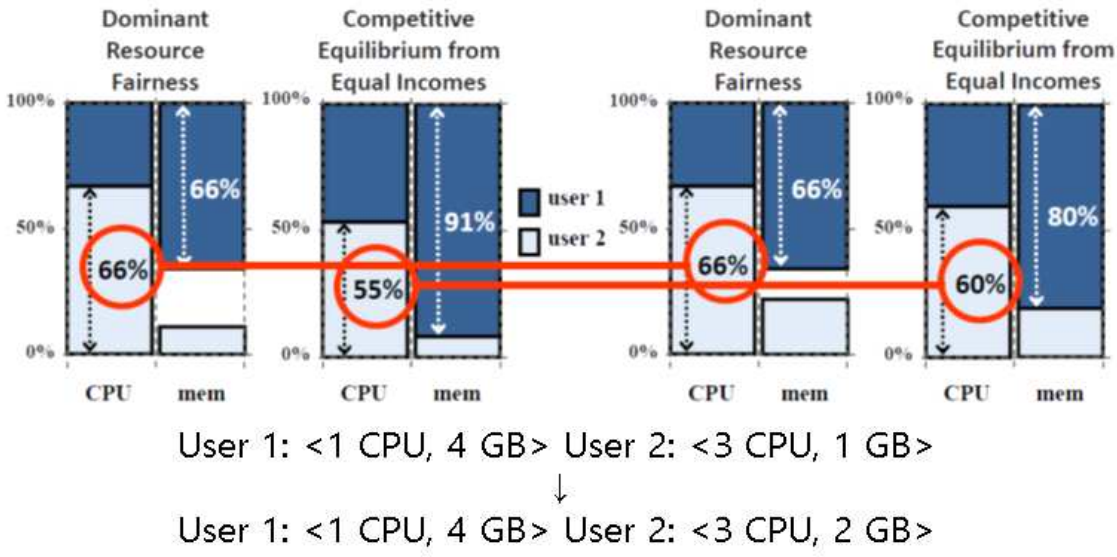

그림 5. DRF 알고리즘과 CEEI 알고리즘 비교 [5]

Algorithm 1 DRF pseudo-code

$R = \langle r_1, \cdots, r_m \rangle$ ▷ total resource capacities
$C = \langle c_1, \cdots, c_m \rangle$ ▷ consumed resources, initially 0
$s_i \ (i = 1..n)$ ▷ user i's dominant shares, initially 0
$U_i = \langle u_{i,1}, \cdots, u_{i,m} \rangle \ (i = 1..n)$ ▷ resources given to user i, initially 0

pick user i with lowest dominant share s_i
$D_i \leftarrow$ demand of user i's next task
if $C + D_i \leq R$ **then**
  $C = C + D_i$ ▷ update consumed vector
  $U_i = U_i + D_i$ ▷ update i's allocation vector
  $s_i = \max_{j=1}^{m} \{u_{i,j}/r_j\}$
else
  return ▷ the cluster is full
end if

그림 6. DRF 알고리즘 [5]

o [확인/수정 필요]HDRF는 각 프레임워크의 Job 단위만 커버하는 단점을 보완하고, 또한 Job 별 Task들도 자원사용량이 다를 수 있는 문제를 풀기위해 제안되었다. <그림 7>은HDRF도 세부적으로 ??개의 방법 중에서 ??를 사용한다.

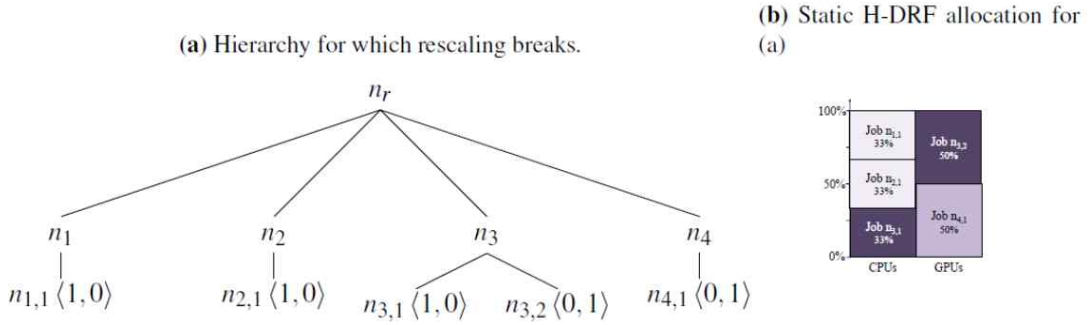

그림 7. HDRF 개념도 [4]

o Mesos Cluster의 자원은 중앙의 Master(s)에 의해서 할당되고 관리되기 때문에 자원관리 관점에서는 중앙집중형, Job 관리 관점에서는 분산형으로 구분할 수 있다. Mesos Framework는 Scheduler와 Executor로 구성된다. Framework Scheduler는 Mesos Master에게 자원을 할당받고 Job Scheduler 로직에 따라 Mesos Slave(s)에 Executor를 실행한다. Mesos의 자원관리는 Linux Kernel 기능인 cgroup과 namespace를 활용한 Linux Container 기반으로 수행된다. Mesos 자체 컨테이너와 Docker [6]컨테이너를 사용할 수 있다. Executor는 수행할 작업들

을 실행하고 관리하는 역할을 담당한다.

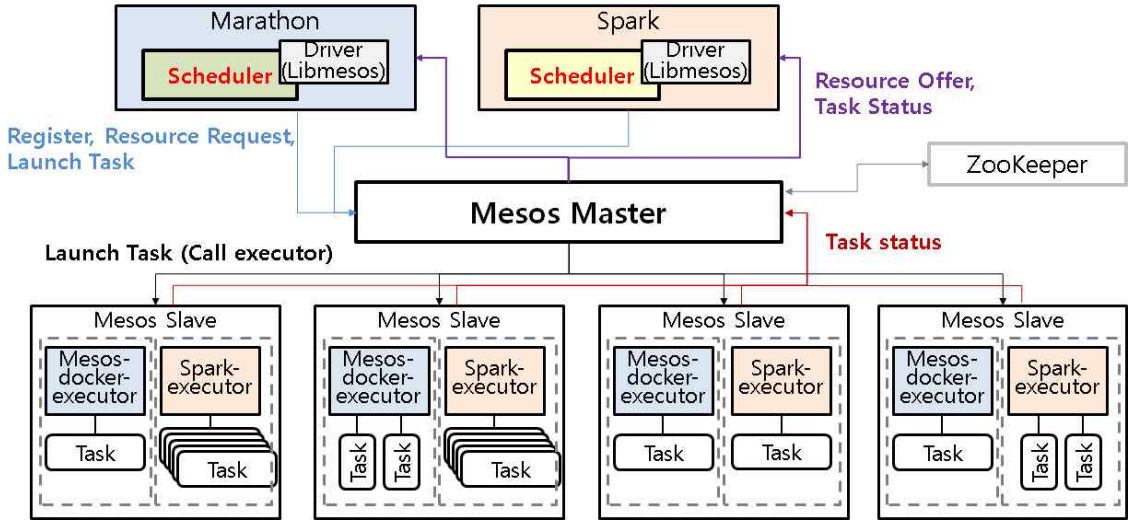

그림 8. Mesos와 Mesos Framework 구성 예

o Framework가 Scheduler와 Executor로 구분되어서 만약 Scheduler 동작, 또는 연결 등 장애가 발생하였을 때 Job 및 Task들은 영향을 받지 않고 실행될 수 있다. 그리고 Scheduler가 복구되었을 경우 Scheduler와 Executor를 다시 연결 (맵핑)할 수 있다.

3.3. Mesos Frameworks

o 대표적인 Mesos의 Framework는 Apache Aurora (community) [7], Mesosphere의 Marathon [8], Chronos [9] 그리고 Apache Spark [10]가 있다. Aurora는 Marathon과 함께 Mesos의 대표적인 Framework로 Twitter 엔지니어 중심으로 개발된 오픈소스 프로젝트이자 커뮤니티이다. 응용과 서비스들을 클러스터 위에서 운영(프로덕션 레벨)하기 위한 것으로 실행하는 job들에 대한 롤링 업데이트+롤백, 스케일링, 안정성, 보안에 초점을 맞추었으나. Marathon에 비해 사용하기 어려울 수 있다. Domain Sepecific Language (DSL)을 활용하는 Aurora API 사용하는 것이 특징이다.

o Marathon은 Mesosphere 소유의 오픈소스로 Mesosphere DCOS Container Orchestration Engine (COE). Aurora와 Marathon 모두 native Docker 지원한다. Marathon은 사용이 쉽고 단순하게 JSON을 활용한 REST API 사용하지만 Production 레벨에서는 Marathon + Chronos 형태로 활용함. Docker는 Aurora를 Docker Swarm으로 운영되는 클러스터의 COE로 개발하여 Docker쪽 생태계를 키울 것으로 보인다.

4. Mesos를 활용한 Spark 운용사례

4.1. Apache Spark 소개

o Apache Spark는 in-memory based analytic engine을 지향하는 오픈소스 소프트웨어이다. 대다수의 Map-Reduce Job들이 작업을 반복하는 특징이 있고, 데이터의 입출력은 스토리지기반의 HDFS를 활용한다. Spark는 이러한 데이터 입출력 구조를 메모리를 활용하도록 설계하여 속도를 빠르게 하면서, Hadoop 보다 더욱 일반화 된 Analytics Engine를 지향한다.

o Spark Core를 기반으로 실시간성 데이터를 처리하기 위한 Streaming, 많이 사용하는 머신러닝 알고리즘을 지원하는 Mllib, Legacy DB와의 통합을 위한 SparkSQL 등 다양한 확장이 존재한다.

o Spark 클러스터를 구성하는 방법은 3가지가 있다. 첫째, Standalone 모드는 Spark 자체적으로 클러스터를 구성한다. 두 번째, Mesos 모드는 클러스터 메니저로 Mesos를 활용하는 것으로 Spark가 Mesos의 Framework로 동작한다. 세 번째, YARN 모드는 Hadoop 클러스터 메니저인 YARN을 활용한다. Hadoop을 중심으로 운용되는 클러스터에서 Spark와 상호보완하는 Workload를 운용할 때 유용하다.

o Spark는 Mesos 클러스터에 작동할 때 Job Scheduling 모드로 Fine-grained와 Coarse-grained 중에 선택할 수 있다. Coarse-grained 모드는 Spark의 Job 단위로 Mesos가 offer하는 자원을 최대한 확보하여 스케줄링하고, Fine-grained 모드는 Spark의 Task별로 Mesos가 offer하는 자원을 할당한다. 클러스터의 가용자원양이 충분할 때에는 Coarse-grained 모드가 스케줄링 오버헤드가 없으므로 성능이 좋지만, 충분하지 않다면 필요한 자원이 확보되기 까지 대기하므로 이때에는 Fine-grained 모드가 성능이 좋다.

4.2. Spark-Perf

o 아직 Spark를 활용한 분석모델과 서비스를 개발하지 않았으므로, Spark의 작동원리와 성능 등을 테스트하기 위해 벤치마크 툴인 Spark-Perf [11]를 사용하였다. Spark-Perf는 Spark를 상용 서비스하는 회사인 Databrix에서 개발한 오픈소스로 Spark Core의 성능측정과 Spark Machine Learning Library (Mllib)를 활용하는 워크로드의 성능측정을 할 수 있다.

o Spark-Perf에서 워크로드는 Scale-Factor로 정의할 수 있으며, Scale-Factor 1은 20대의 Amazon EC2 xlarge 클러스터와 같다. 참고로 m1.xlarge의 리소스는 CPU 4

core, RAM 15 GB, Storage 1680 GB (4x420GB)이다.

o Spark-Perf로 벤치마킹 할 수 있는 머신러닝 알고리즘들은 다음과 같다.
- glm-regression: Generalized Linear Regression Model
- glm-classification: Generalized Linear Classification Model
- naive-bayes: Naive Bayes
- naive-bayes-bernoulli: Bernoulli Naive Bayes
- decision-tree: Decision Tree
- als: Alternating Least Squares
- kmeans: K-Means clustering
- gmm: Gaussian Mixture Model
- svd: Singular Value Decomposition
- pca: Principal Component Analysis
- summary-statistics: Summary Statistics (min, max, ...)
- block-matrix-mult: Matrix Multiplication
- pearson: Pearson's Correlation
- spearman: Spearman's Correlation
- chi-sq-feature/gof/mat: Chi-square Tests
- word2vec: Word2Vec distributed presentation of words
- fp-growth: FP-growth frequent item sets

4.3. Mesos Cluster에서 Spark-Perf를 활용한 실험

o Spark-Perf로 벤치마킹할 수 있는 Mllib 알고리즘 중에서 컴퓨팅 자원을 많이 사용하는 Unsupervised Learning 기법인 K-means를 대상으로 실험을 수행하였다.

o <그림 9>는 전체 실험절차를 나타낸 것으로, 임의의 데이터를 생성하고 Mllib를 호출하는 기능은 Scala로 구현되어 있다. 또한 실험의 변수를 설정하는 부분은 Python으로 구현되어 있다.

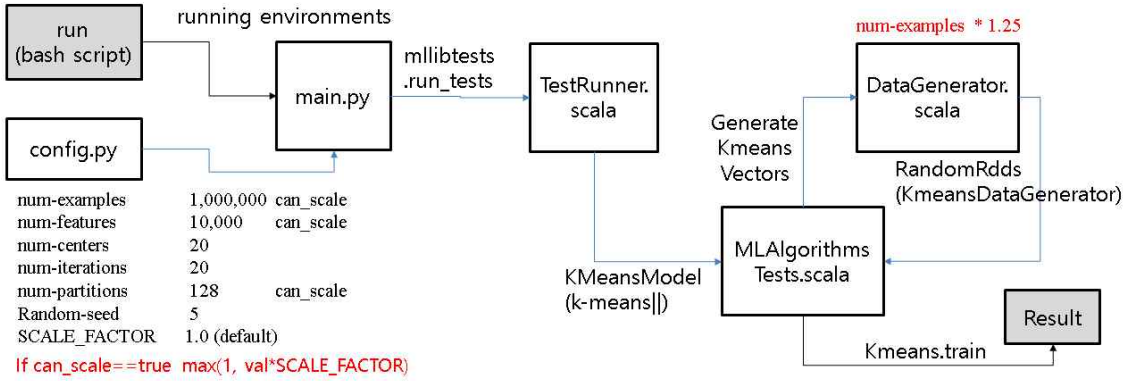

그림 9. Spark-Perf의 K-means 벤치마킹 Workflow

o SmartX Type-M Playground의 Worker 노드를 2대로 구성하여, Workload (Scale-Factor)를 각각 0.001, 0.01, 0.1, 0.5, 1로 설정하여 실험을 한 결과는 <그림 10>과 같다. Scale-Factor가 0.1일 때 10배 작은 0.01보다 오히려 성능이 좋았다. 이 경우 충분히 크지 않은 워크로드는 병렬화에 따른 오버헤드가 더 클 수 있음을 보여준다. 그리고 Spark만 단독으로 실행하는 상황에서 Coarse-grained 스케줄링이 성능이 더 좋았다. Fine-grained와의 성능차이를 스케줄링 오버헤드로 볼 수 있다.

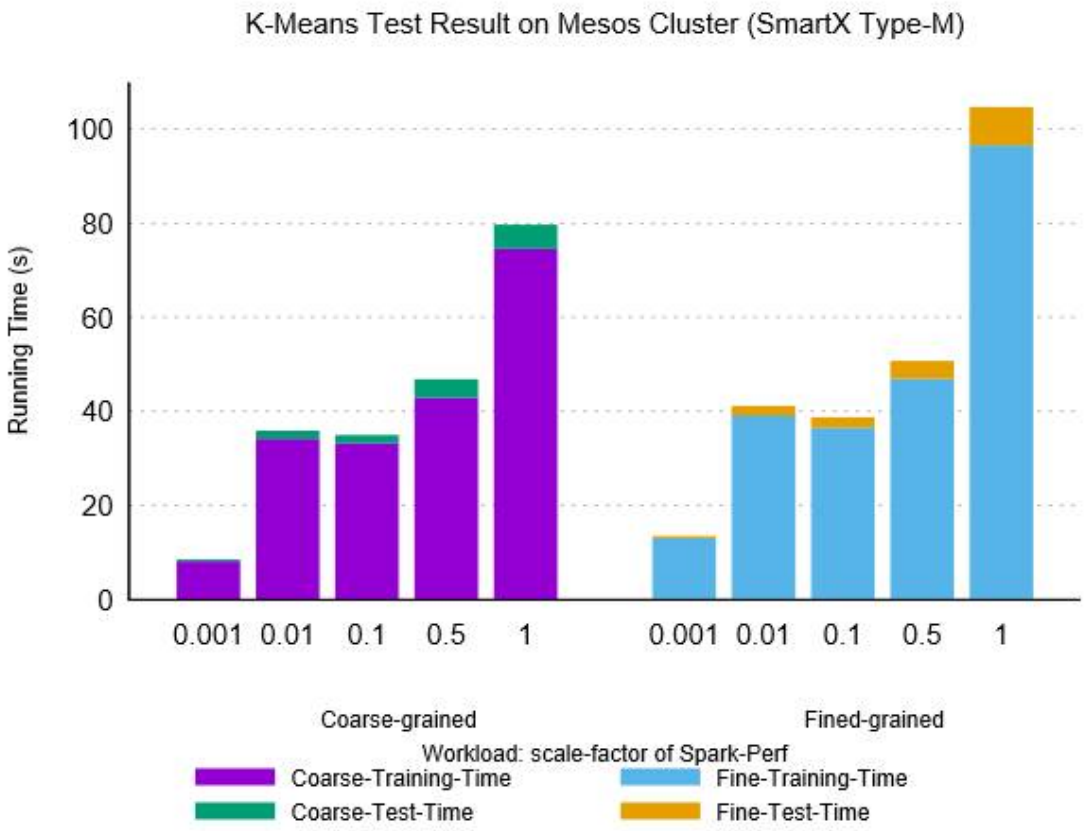

그림 10. Workload별 Spark-Perf K-means 테스트 결과

5. 결론 및 향후계획

o OpenStack의 가상화 기반 Compute Node의 오버헤드를 줄이는 방법 중 하나인 Ironic Bere-metal service에 대해 간략히 정리하고, 클러스터 자원을 관리하는 소프트웨어 중 Mesos에 대한 구조와 작동원리를 기술하였다. 필요시 Mesos의 자원 할당 모듈을 따로 구현하여 클러스터 사용자의 요구사항에 맞게 커스터마이징 할 수 있으며, 각 Framework의 측면에서는 클러스터의 자원관리에 대한 고민없이 Mesos를 활용하여 Job 스케줄링을 할 수 있다.

o Data Analytic engine 중 하나인 Spark를 기반으로 벤치마킹 프로그램인 Spark-Perf를 실행하였고, 그 중에서 K-means 테스트를 선정하여 Spark-Perf의 동작과 Spark의 Job 스케줄링 방법에 따른 성능차이를 알아보았다.

o 향후 Ironic 이미지를 만들거나 관리하는 방법, 2~3개 정도의 여러 개 Framework가 동시에 Mesos Cluster에서 실행되는 환경, BigData/HPC 응용들을 Linux Containerize 하는 것에 대해 고려하여 본 기술문서의 내용을 보완할 것이다.

References

[1] OpenStack, http://openstack.org.
[2] Ironic, https://wiki.openstack.org/wiki/Ironic.
[3] Mesos, http://mesos.apache.org.
[4] ZooKeeper, http://zookeeper.apahe.org.
[5] Bhattacharya, Arka A., et al. "Hierarchical scheduling for diverse datacenter workloads." Proceedings of the 4th annual Symposium on Cloud Computing. ACM, 2013.
[6] Ghodsi, Ali, et al. "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types," in Proc. NSDI. Vol. 11. 2011.
[7] Docker, http://docker.io.
[8] Aurora, http://aurora.apache.org.
[9] Marathon, https://mesosphere.github.io/marathon.
[10] Chronons, https://mesos.github.io/chronos.
[11] Spark, http://spark.apache.org.
[12] Spark-Perf, https://github.com/databricks/spark-perf.

K-ONE 기술 문서

- 이 문서의 기술적인 내용은 프로젝트의 진행과 함께 별도의 예고 없이 변경될 수 있습니다.

- 본 문서와 관련된 문의 사항은 아래의 정보를 참조하시길 바랍니다.
  (Homepage: http://opennetworking.kr/projects/k-one-collaboration-project/wiki, E-mail: k1@opennetworking.kr)

작성기관: K-ONE Consortium
작성년월: 2016/05