최종보고서

슈퍼컴퓨터 운영 관리 시스템 구축을 위한 Intel RSD 기반 하드웨어 관리 및 모니터링 기술 분석 보고서

2016. 11.

부 서: 슈퍼컴퓨터시스템개발실
작성자: 박주원, 조혜영, 박주연

연구기관 : 한국과학기술정보연구원

Korea Institute of Science and Technology Information

목 차

표 차 례

그 림 차 례

제 1 장 Intel RSD 기반 하드웨어 관리

제 1 절 Intel RSD

1. Intel RSD 소개

일반적으로 서버는 CPU, 메모리, 디스크, NIC 카드 등 다양한 디바이스로 구성되며 제작된 형태에 따라 다음 종류로 구분해 볼 수 있다.

□ Blade Server

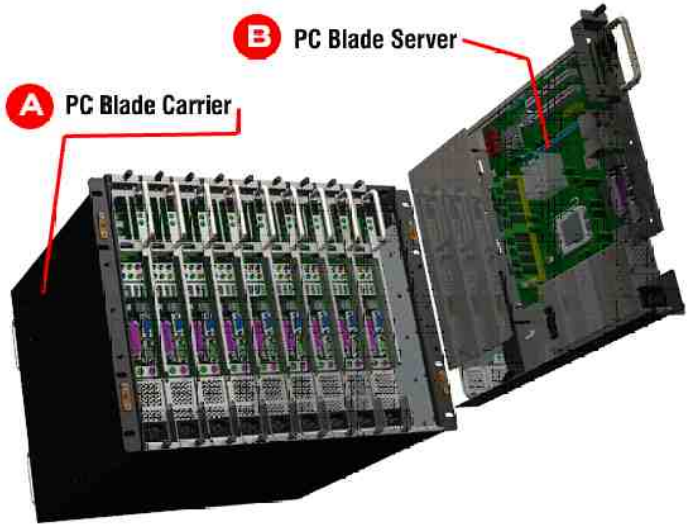


그림 1. 블레이드 서버

블레이드 서버는 웹서비스와 같은 단일 작업 실행에 전념할 수 있도록, 하나 이상의 CPU와 메모리를 담고 있는 얇고 모듈화된 전자 기판을 통해 제작된 서버이다. 보통 하나의 캐리어에 다수의 블레이드 서버를 패키지함으로써 집적도를 높일 수 있다.

□ Micro Server

그림 2. 마이크로 서버

저전력 x86 CPU Atom과 서버용 ARM 프로세서를 기반으로 한 작고 저렴하며 소비전력이 낮은 형태로 제작된 서버를 말한다. 마이크로서버는 출현 초기는 CPU 부하가 적은 웹 서비스에 한정되었다면 최근에는 64비트 지원 CPU의 출현과 메모리 용량 증가로 동적 웹 콘텐츠 처리, 데스크톱 호스팅, 신호처리 등 다양한 업무에도 사용되고 있으며 점차 그 영역이 확대될 것으로 기대된다[1].

2013년 인텔은 데이터 센터의 서버, 네트워크, 스토리지등 물리적인 하드웨어 자원을 추상화 (abstract)하여 소프트웨어 기반으로 자원을 제어하기 위해 "Software Defined Infrastructure (SDI)"를 발표했다. 이는 RSD (Rack Scale Design)라는 하드웨어 규격을 제시하고 규격에 적합한 자원은 소프트웨어 기반으로 추상화된 자원의 풀을 생성하여 사용자의 요구에 따라 동적으로 구성하여 제공한다.

<그림3>에서도 확인할 수 있듯이 현재는 하나의 랙에 다수의 랙마운트 서버를 장착하여 전력 및 냉각 장치를 공유하는 형태로 나타나고 있으나 최근에는 디바이스간의 연결성을 제공하기 위해 실리콘포토닉스와 같은 광 통신 기술을 활용하여 랙 내에서의 고속의 인터커넥션을 제공하기 위한 네트워크 기술이 발달하고 있다. 이러한 기술을 기반으로 향후에는 랙에 있는 모든 하드웨어를 추상화하여 컴퓨팅, 스토리지, 메모리등 각 모듈별로 자원의 풀(pool)을 구성하고 사용자의 요청에 따라 동적으로 자원 풀에서 필요한 만큼만 자원을 할당 받아 서버를 구성하여 제공한다. 이를 위해 인텔에서는 <그림4>과 같은 참조 아키텍쳐를 제공하고 있다. 이를 통해 아래와 같은 장점을 제공할 수 있다.

① 소프트웨어로 구성 가능한 시스템과 어플리케이션을 위해서 신속한 프로비저

닝을 제공하며, 이를 통해 서비스 제공 속도를 높입니다.

② 자원 활용과 상호 운용성을 개선하여 운영 효율성을 극대화합니다.

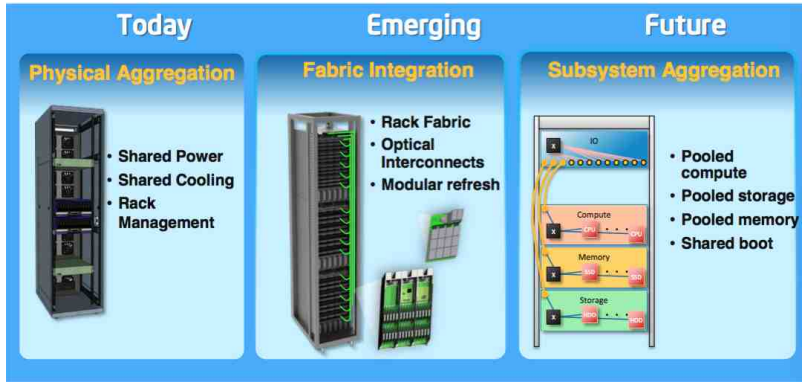


그림 3. 하드웨어 기술 동향 (인텔 자료) source: Data Center & Connected Systems Group, "Intel Rack Scale Architecture Overview," 2013-05-09.

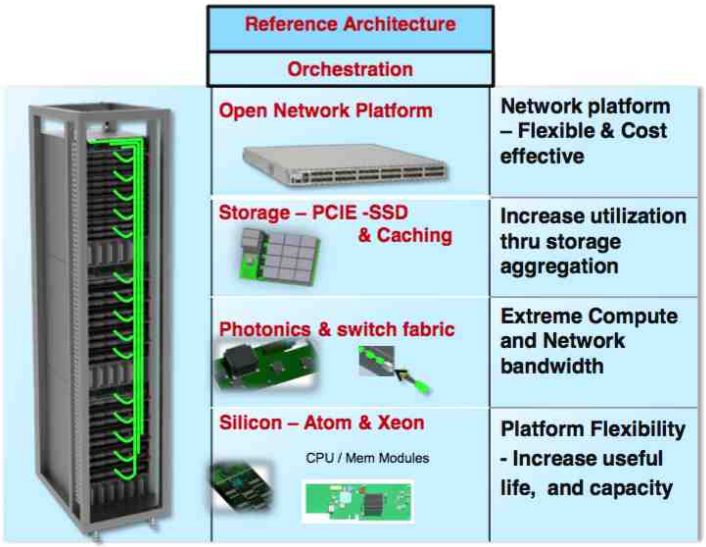


그림 4. 하드웨어 기술 동향 (인텔 자료) source: Data Center & Connected Systems Group, "Intel Rack Scale Architecture Overview," 2013-05-09.

2. 아키텍쳐

인텔 RSD의 계층 구소는 아래의 그림과 같다. 관리 대상은 Blade, Module, Drawer, Rack, Pod로 구분하고 있으며 이를 BMC (Baseboard Management Controller), MMC (Module Management Controller), PSME(Pooled system management engine), RMM (Rack Management Module), PODM(POD Manager) 등 핵심 요소들이 연동되어 자원을 관리하고 있다.

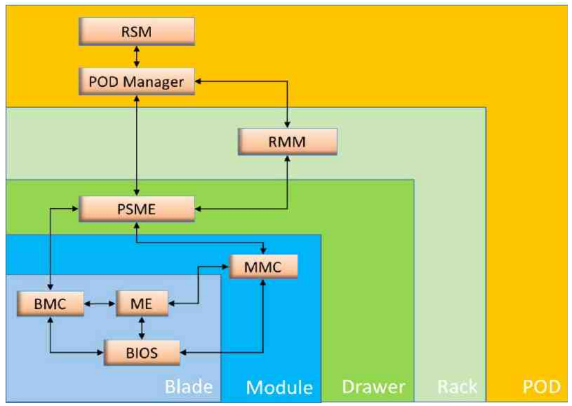


그림 5. Pod Logical Hierarchy. Source: Intel Rack Scale Design. Architectural Requirements Specification. 2016-08.

자원을 효율적으로 관리하고 문제가 되는 모듈을 정확히 파악하기 위해서는 자원 식별 및 위치 파악이 매우 중요하다. 인텔 RSD에서의 자원 식별 순서는 대부분 '좌 → 우' & '위 →아래' 순으로 진행되며, 그 반대의 경우도 존재한다.

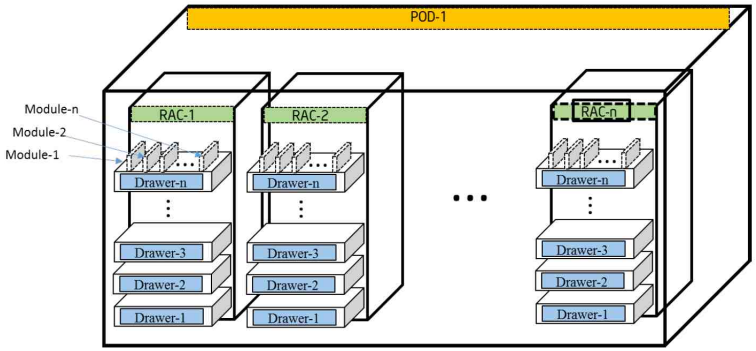


그림 6. Intel Rack Scale Design component location identification. Source: Intel Rack Scale Design. Architectural Requirements Specification. 2016-08.

3. Intel RSD 구성 요소

인텔 RSD는 컴퓨팅, 패브릭, 스토리지 및 관리 모듈이 유기적으로 연동되어 광범위한 가상 시스템을 구축한다. 이는 다음과 같은 아래 4가지의 구성 요소를 토대로 구축된다.

□ **Pod Manager** : 멀티 랙을 관리합니다. 하드웨어 아랫단과 오케스트레이션 계층 위단 위치하며, 자원과 정책을 관리하고 표준 인터페이스를 통해 구현되는 펌웨어이자 소프트웨어 API입니다.

□ **Pooled System** : 컴퓨팅, 네트워크 및 스토리지 자원 풀(pool)을 워크로드 요구사항에 따라 구성하는 시스템입니다.

□ **Pod-wide storage** : Pod 규모 스토리지(이더넷으로 연결된 스토리지 기반)에 멀티 랙 자원 또는 스토리지 하드웨어 및 컴퓨팅 노드(로컬 스토리지 포함)로 구축되며, 다양한 활용이 가능한 스토리지 알고리즘이 사용됩니다.

□ **Configurable network fabric** : 최신 TOR(Top-Of-Rack) 스위치 설계와 플랫폼을 통한 분산 스위치 등 경제적인 네트워크 토폴로지를 다양하게 지원합니다.

4. Intel RSD 주요 API

□ **PSME REST API**

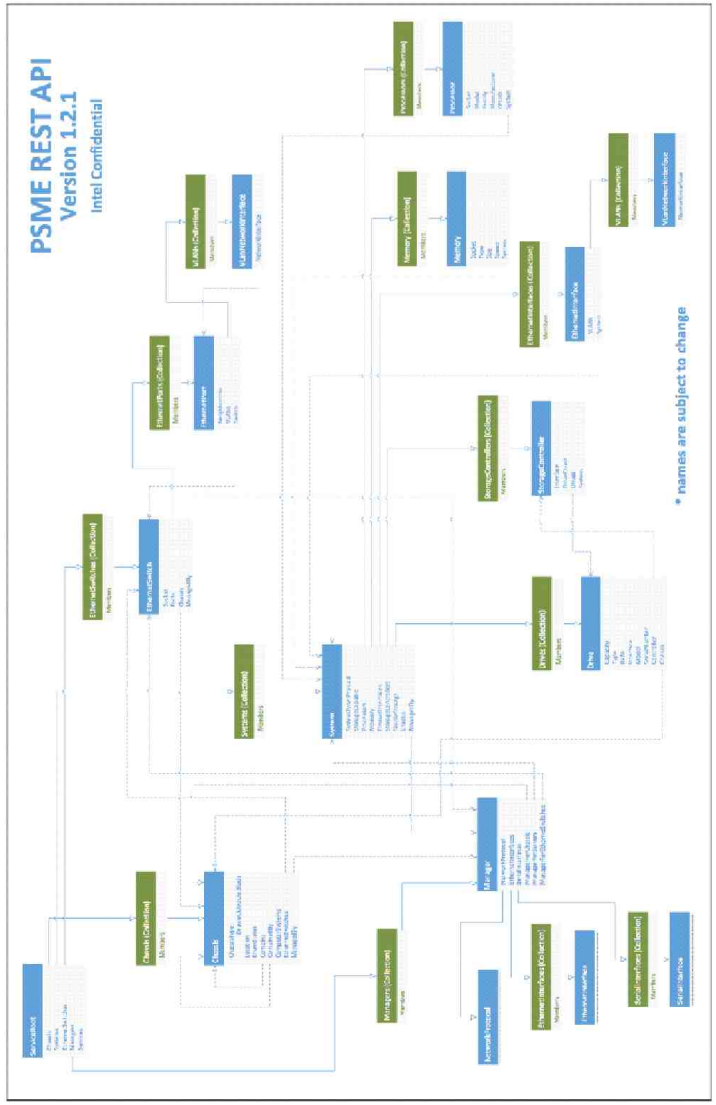


그림 7. PSME REST API hierarchy. Source: Intel Rack Scale Design PSME REST - API Sepcification. 2016-09

Resource	URI
Service Root	/rest/v1
Drawer Collection	/rest/v1/Drawers
Drawer	/rest/v1/Drawers/{drawerID}
Comput e Module Collecti on	/rest/v1/Drawers/{drawerID}/ComputeModules
Compute Module	/rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}
Blade Collection	/rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades
Blade	/rest/v1/Drawers/{drawerID}//ComputeModules/{moduleID}/Blades/{bladeID}
Processors Collection	/rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/Proc essors
Processor	/rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/Proce ssors/{processorID}
Memory Collection	/rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/Mem ory
Memory	/rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/Mem ory/{memoryID}
Storage Controllers Collection	/rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/Stora geControllers
Storage Controller	/rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/Stora geControllers/ {controllerID}
Drives Collection	/rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/Stora geControllers/ {controllerID}/Drives
Drive	/rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/Stora geControllers/ {controllerID}/Drives/{driveID}
Manager collection	/rest/v1/Managers
Manager	/rest/v1/Managers/{managerID}

Network Service	/rest/v1/Managers/{managerID}/NetworkService
Network Interface Collection	/rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/Ether netInterfaces /rest/v1/Managers/{managerID}/EthernetInterfaces
Network Interface	/rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/Ether netInterfaces/{nicID} /rest/v1/Managers/{managerID}/EthernetInterfaces/{nicID}
Fabric Module Collection	/rest/v1/Drawers/{drawerID}/FabricModules
Fabric Module	/rest/v1/Drawers/{drawerID}/FabricModules/{moduleID}
Fabric Switch Collection	/rest/v1/Drawers/{drawerID}/FabricModules/{moduleID}/Switches
Fabric Switch	/rest/v1/Drawers/{drawerID}/FabricModules/{moduleID}/Switches/{switchID}
Fabric Switch Port Collection	/rest/v1/Drawers/{drawerID}/FabricModules/{moduleID}/Switches/{switchID}/Ports
Fabric Switch Port	/rest/v1/Drawers/{drawerID}/FabricModules/{moduleID}/Switches/{switchID}/Ports /{portID}
VLAN Network Interface Collection	/rest/v1/Drawers/{drawerID}/FabricModules/{moduleID}/Switches/{switchID}/Ports /{portID}/VLANs /rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/Ether netInterfaces/{nicID} /VLANs
VLAN Network Interface	/rest/v1/Drawers/{drawerID}/FabricModules/{moduleID}/Switches/{switchID}/Ports /{portID}/VLANs/ {vlanID} /rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/Ether netInterfaces/ {nicID}VLANs/{vlanID}

표 1. Resource and URLs. Source: Intel Rack Scale Design PSME REST - API Sepcification. 2016-09

□ PODM REST API

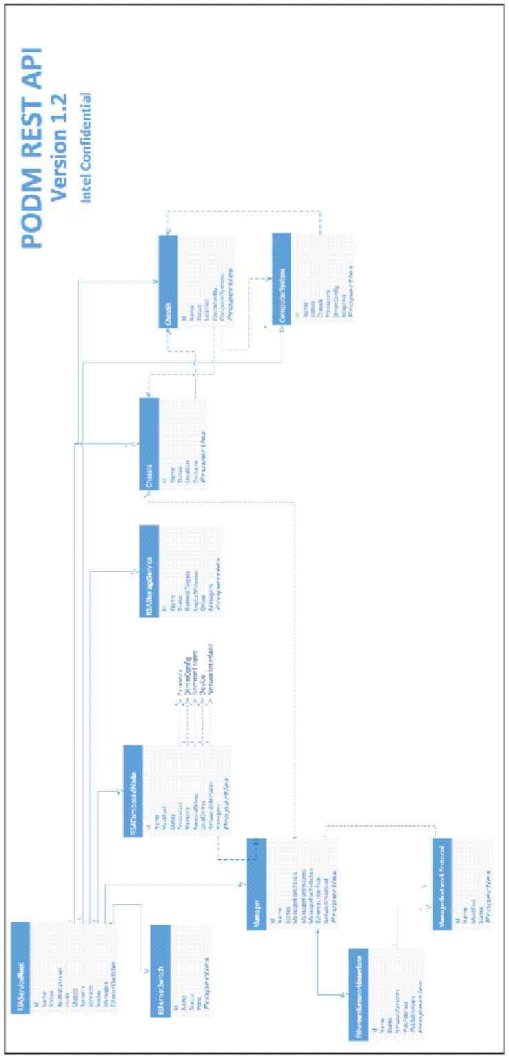


그림 8. POD Manager REST API hierarchy. Source: Intel Rack Scale Design POD Manager - API Sepcification. 2016-09

Resource	URI
Service Root	/rest/v1
PodCollection	/rest/v1/Pods
POD	/rest/v1/Pods/{podID}
Rack Collection	/rest/v1/ Pods/{podID}/Racks
Rack	/rest/v1/Pods/{podID}/Racks/{rackID}
Drawer Collection	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers
Drawer	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}
Compute Module Collection	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules
Compute Module	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}
Blade Collection	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}/ Blades
Blade	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}/ Blades/{bladeID}
Processor Collection	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}/ Blades/{bladeID}/Processors
Processor	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}/ Blades/{bladeID}/Processors/{processorID}
Memory collection	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}/ Blades/{bladeID}/Memory
Memory	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}/ Blades/{bladeID}/Memory/{memoryID}
Storage controller collection	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}/ Blades/{bladeID}/ StorageControllers
Storage controller	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}/ Blades/{bladeID}/StorageControllers/{controllerID}
Drive collection	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}/ Blades/{bladeID}/ StorageControllers/{controllerID}/Drives
Drive	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}/ Blades/{bladeID}/StorageControllers/{controllerID}/Drives/{driveID}
Manager Collection	/rest/v1/Managers
Manager	/rest/v1/Managers/{managerID}
Network service	/rest/v1/Managers/{managerID}/NetworkService
Network interface collection	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}/ Blades/{bladeID}/EthernetInterfaces /rest/v1/Managers/{managerID}/EthernetInterfaces

Network interface	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerID}/ComputeModules/{moduleID}/ Blades/{bladeID}/EthernetInterfaces/{nicID} /rest/v1/Managers/{managerID}/EthernetInterfaces/{nicID}
Fabric Module Collection	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerId}/FabricModules
Fabric Module	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerId}/FabricModules/{moduleID}
Fabric Switch Collection	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerId}/FabricModules/{moduleID}/Sw itches
Fabric Switch	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerId}/FabricModules/{moduleID}/Sw itches/{switchID}
Fabric Switch Port Collection	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerId}/FabricModules/{moduleID}/Sw itches/{switchID}/Ports
Fabric Switch Port	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerId}/FabricModules/{moduleID}/Sw itches/{switchID}/Ports/{portID}
VLAN network interface collection	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerId}/FabricModules/{moduleID}/Sw itches/{switchID}/Ports/{portID}/VLANs /rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/EthernetInt erfaces/{nicID}//VLANs
VLAN network interface	/rest/v1/Pods/{podID}/Racks/{rackID}/Drawers/{drawerId}/FabricModules/{moduleID}/Sw itches/{switchID}/Ports/{portID}/VLANs/{vlanID} /rest/v1/Drawers/{drawerID}/ComputeModules/{moduleID}/Blades/{bladeID}/EthernetInt erfaces/{nicID}VLANs/{vlanID}
Storage Service Collection	/rest/v1/Services
Storage Service	/rest/v1/Services/{serviceID}
Remote Target Collection	/rest/v1/Services/{serviceID}/Targets
Remote target	/rest/v1/Services/{serviceID}/Targets/{targetID}
Logical Drive Collection	/rest/v1/Services/{serviceID}/LogicalDrives
Logical Drive	/rest/v1/Services/{serviceID}/LogicalDrives/{driveID}
Physical Drive Collection	/rest/v1/Services/{serviceID}/Drives
Physical Drive	/rest/v1/Services/{serviceID}/Drives/{driveID}
Composed Node Collection	/rest/v1/Systems
Composed Node	/rest/v1/Systems/{nodeID}

표 2. Resources and URLs. Source: Intel Rack Scale Design POD Manager - API Sepcification. 2016-09

제 2 장 모니터링 기술 분석

제 1 절 네트워크 모니터링 기법

1. 측정 지표

□ **지연 (latency):** 일반적으로 지연은 송신시스템에서 전송한 패킷이 수신 시스템에 도착할 때 까지 걸리는 시간으로 볼 수 있다. 이를 좀 더 세분화 하면 1) 접근 지연, 2) 큐 지연, 3) 전송 지연 (Propagation delay) 으로 나누어 볼 수 있다. 접근 지연은 송신 시스템에서 정보를 보자고자 할 때 전송 매체가 가용하거나 네트워크가 전송 준비가 될 때 까지 송신 시스템이 기다리는 시간을 말한다. 큐 지연은 라우터에서 경로 결정 시 큐에서 기다리는 시간을 말한다. 일반적으로 정보를 전송하기 위해서 송신 시스템에서 정보를 패킷이라는 형태로 만들어 디폴트 라우터 또는 게이트웨이 (default 라우터)로 전송하고 라우터는 받은 패킷의 헤더 정보를 이용하여 다음 라우터를 결정한다. 이렇게 다수의 라우터를 통해 최종 목적지까지 도달하게 된다. 이때 라우터는 다양한 경로에서 오는 패킷을 순차적으로 처리하기 위해 일단 큐에 패킷을 넣고 하나씩 꺼내어 처리하는데 이때 큐에서 기다리는 시간을 큐 지연이라 한다. 전송 지연 (propagation delay(은 일정한 거리에 위치한 라우터 사이의 패킷 전송시 발생하는 지연을 말한다. 이 모든 지연을 합하여 송신 측에서 전송한 정보가 수신측에 수신되는 될 때까지의 모든 지연을 합하여 종단간 지연 (end-to-end latency)라고 한다.

□ **지터 (jitter):** 지터(jitter)는 일반적으로 종단간의 지연의 분산을 의미한다. 모든 네트워크는 물리적 지터가 발생한다. 가령 신호를 재형성하는 기능을 하는 리피터(repeater)가 가끔씩 비정상적인 작동을 할 수도 있고 금속도체의 경우 온동의 변화에 따라 전파지연(propagation delay)등이 변할 수 있기 때문에 물리적 지터가 존재하는 것이다. 특히, 멀티미디어를 전송하고 재생하기 위해서는 지터에 대한 제어가 매우 중요하다. 멀티미디어는 데이터를 재생하는 시간이 있기 때문에 지터가 없는 경우에는 재생하는 시간 차이가 있지만 안정적으로 플레이가 가능하다. 그러나 지터가 큰 경우에는 멀티미디어 재생 도중에 끊어짐 현상이 발생한다. 이러한 지터의 영향을 줄이기 위해 수신측에서는 일정한 크기의 버퍼를 생성하여 수신된 패킷을 저장한 후 재생 시간에 맞게 멀티미디어를 재생한다.

□ **손실률(loss):** 에러는 데이터의 변경, 손실, 중복, 비순서적인 전송 등에 의해 일어난다. 네트워크 에러는 비트 에러율(bir error rate), 패킷/셀 에러율(packet/cell error rate), 패킷/셀 손실율(packet/cell loss rate) 등으로 구분할

수 있다. 일반적으로 송신 시스템에서는 에러 발생 여부를 감지하는 데이터 무결성을 검사하기 위해 CRC (Cyclic Redundancy Check) 비트를 추가하여 전송하고 수신 시스템에서는 CRC 비트를 이용하여 에러를 감지한다.

□ **대역폭 (Bandwidth):** 네트워크에서 처리 가능한 비트수로 처리율(Throughput) 또는 전송율 (transfer rate)라고도 부른다. 대역폭은 Capacity, Available bandwidth로 구분할 수 있다. Capacity는 네트워크의 최대 처리량을 의미한다. 가용 대역폭은 현재 사용할 수 있는 대역폭으로 Capacity에서 현재 사용량을 제외한 값이다.

2. 능동 모니터링 기법

□ 능동 모니터링 방식은 측정을 위해서 별도의 측정 패킷 (Probe Packet)을 이용하는 방식이다. 주로 가용 대역폭 (available bandwidth), 병목(bottleneck) 지점 검출, 그리고 특정경로상의 지연, 지터, 손실률등을 측정하는데 주로 이용된다. 대표적인 능동 모니터링 방식으로는 ICMP 패킷을 활용하는 Ping과 TraceRoute가 있다. 능동 모니터링 방식은 측정 패킷의 크기가 작고 분석해야 할 패킷의 양이 적기 때문에 측정 패킷을 받아서 분석하는 시스템의 부하가 크지 않다. 반면, 실제 사용자 트래픽을 측정하는 것이 아닌 경로의 상태를 측정하기 때문에 사용자가 체감하는 네트워크 상태와 차이가 있을 수 있다. 또한 네트워크가 용대역폭이 낮은 경우에는 측정 데이터가 사용자 데이터와 경쟁하기 때문에 네트워크 상태를 더욱 악화시키는 문제점이 있다.

3. 수동 모니터링 기법

□ 수동 모니터링 방식은 별도의 측정 패킷을 만들지 않고 사용자 트래픽을 이용하여 네트워크 상태를 모니터링 하는 방식으로 사용자 트래픽의 전송율이나 데이터 플로우의 지연, 지터, 손실률과 같은 전송 상태를 측정하는데 적합하다. 대표적인 수동 모니터링 방식으로는 탭(Tap)이나 미러링(Mirroring)을 이용하여 사용자 트래픽을 모니터링 하는 방법이 있다. 수동 모니터링 방식은 별도의 측정 패킷을 전송하기 않기 때문에 네트워크의 부하 증가 없이 사용자의 데이터 플로우를 측정할 수 있다는 장점이 있지만 측정해야 할 패킷의 크기가 크고 패킷의 수가 많기 때문에 분석하는 시스템의 부하가 커질 수 있다. 또한 사용자 트래픽이 송수신 되는 동안에만 측정이 가능하기 때문에 경로의 상태를 측정하기에는 부적합하다.

4. 혼성 모니터링 기법

□ 최근 들어 측정 성능을 증대시키기 위해서 능동 모니터링 방식과 수동 모니터링 방식을 혼용하여 사용하려는 노력이 확대되고 있다[2][3]. [2]에서는 그리드 상에서 확장성 있는 측정을 위해서 그리드 응용 프로그램이 실행되고 있는 동안에는 응용 프 로그램의 패킷을 활용한 수동 모니터링 방식을 사 용하고, 응용 프로그램이 실행되지 않는 동안에만 능동 모니터링 방식을 이용하는 방법을 제안하였다. [3]에서는 종단간 성능 향상을 위해서 능동 모니터 링 방식과 수동 모니터링 방식을 통해 측정된 결과 를 비교, 분석하여 성능 저하의 원인을 판단하고 제 거함으로써 종단간 성능 향상시키기 위한 혼성 모니터링 방식을 연구하였다.

제 2 절 컴퓨팅 모니터링 기법

1. 측정 지표

□ **온도/팬속도:** 일반적으로 시스템의 온도는 시스템의 수명 및 성능에 많은 영향을 미치므로 매우 중요한 측정 지표이다. 그러므로 서버 보드에는 온도를 감지하고 제어할 수 있도록 센서가 부착되어 있으며 팬 속도를 제어함으로써 온도를 조절할 수 있다. 이를 위한 온도 및 팬 속도 측정 및 제어는 BMC (Baseboard Management Controller)를 통해 데이터 접근 및 설정이 가능하다.

□ **소비 전력**: 최근 저전력에 대한 요구사항이 증가함에 따라 소비되는 전력량을 줄이고 성능을 향상시키기 위한 연구가 많이 진행되고 있다. 이를 위해서는 메인 보드에서 소비되는 전력을 정확히 측정해야 한다.

□ **CPU 및 메모리 사용량:** 시스템 자원의 이용 상태를 확인할 수 있는 가장 기본적인 정보로 리눅스의 경우 Top 명령어를 통해 확인이 가능하다.

□ **디스크 사용량:** 시스템을 구성하는 스토리지의 사용량을 나타내는 지표로 전체 디스크 용량, 현재 사용량, 가용 디스크 용량등을 확인할 수 있다.

□ **Syslog 이벤트 메시지:** Syslog 이벤트 메시지는 시스로그데몬(syslogd)에서 출력되는 로그로 [4]에서 정의된 형태로 출력된다. Syslog에는 결함의 등급을 표기하는 Severity 값뿐만 아니라 결함의 원인을 유추할 수 있는 메시지를 포함하고 있어 결함의 원인을 파악하기 위해 관리자는 보통 Syslog를 분석한다.

□ **작업 현황:** 사용자의 작업의 상태를 나타내는 지표로 스케줄러에서 출력된 로그 파일을 분석하여 측정할 수 있다. 대부분의 스케줄러 로그에는 작업의 제출 시간, 작업별 자원 할당량, 대기 시간, 및 작업 종료 시간등이 포함되어 있어 작업 통계를 구할 수 있다. 스케줄에 따라 로그를 출력하는 항목, 방식이 서로 다르기 때문에 [5]에서는 표준화된 워크로드 형식으로 아래의 항목을 정의하고 있다.

- ✓ Job Number: 정수로 표기된 작업 번호. 로그파일에 기록된 첫 번째 작업을 1로하여 순차적으로 부여함.
- ✓ Submit Time: 초 단위로 기록된 작업 제출 시각. 로그파일에 기록된 1번 작업의 제출 시간을 0으로 하여 제출된 시간의 차이를 기록.
- ✓ Wait Time: 초 단위로 기록된 큐 대기 시간. 작업 제출 시각과 실제 작업이 실행한 시각의 차이로 기록.
- ✓ Run Time: 초 단위로 기록된 작업 실행 시간. 작업이 시작된 시각과 종료된 시각의 차이로 기록됨.
- ✓ Number of Allocated Processors: 작업에 할당한 프로세서의 수.
- ✓ Average CPU Time Used: 초 단위로 기록된 CPU 사용 시간.
- ✓ Used Memory: KB 단위로 기록된 메모리 사용량.
- ✓ Requested Number of Processors: 사용자에 의해 요청된 프로세스 수.
- ✓ Requested Time: 초 단위로 기록된 사용자가 예측한 작업 실행 시간.
- ✓ Requested Memory: KB 단위로 기록된 사용자에 의해 요청된 메모리 용량.
- ✓ Status: 정수로 기록된 작업 상태 코드 (0~5 값으로 표기).
- ✓ User ID: 사용자 아이디.
- ✓ Group ID: 그룹 아이디.
- ✓ Executable (Application) Number: 동일한 워크 로드에 있는 다수의 응용 프로그램을 구분하기 위해 기록된 값.
- ✓ Queue Number: 하나의 시스템에 다수의 큐가 존재 할 경우 이를 구분하기 위해 기록된 값.

- ✓ Partition Number: 하나의 시스템에 다수의 파티션(partition)이 존재 할 경우 이를 구분하기 위해 기록된 값.
- ✓ Preceding Job Number: 본 작업이 시작하기 전에 종료 되어야하는 이전 작업.
- ✓ Think Time from Preceding Job: 초 단위로 기록된 값으로 이전 작업이 종료 되고 제출된 작업이 시작 할 때까지의 시간.

2. ELK

□ 기술의 정의

ELK Stack은 Elasticsearch, Logstash, Kibana의 약자로서 Elasticsearch는 Apache의 Lucene을 바탕으로 개발한 실시간 분산 검색 엔진이며, Logstash는 각종 로그를 가져와 JSON 형태로 만들어 Elasticsearch로 전송하고 Kibana는 Elasticsearch에 저장된 Data를 사용자에게 Dashboard 형태로 보여주는 Solution이다.

ElasticSearch는 Apache Lucene을 기반으로 하는 분산 검색엔진이지만, 그 자체로 파일를 저장하는 NoSQL 형식의 데이터베이스이다. 따라서 대규모 시스템의 로그와 같이 분산된 시스템의 로그를 수집하여 인덱스를 생성/관리하는데 매우 유용하다.

그림 9. ELK (Elasticsearch, Logstash, Kibana) framework.

- ○ ElasticSearch : Apache의 Lucene을 바탕으로 개발한 실시간 분산 검색 엔진
- ○ Logstash : Logstash는 각종 로그를 가져와 JSON형태로 만들어 ElasticSearch로 전송
- ○ Kibana : Elasticsearch에 저장된 Data를 사용자에게 Dashboard 형태로

보여주는 Solution

□ 기술의 특징

ELK는 설치와 확장이 용이한 오픈소스 기반의 소프트웨어 스택이다. ELK는 다음과 같은 특징을 가지고 있다.

○ 루씬 기반: Elasticsearch는 다국어 검색 지원, 자동 완성 지원, 미리보기 지원등 다양한 기능을 지원하는 루씬을 기반으로 만들어져 유연하고 편리한 인덱싱을 지원한다.

○ 분산 시스템 지원: 유연한 분산 시스템을 지원한다. 특히 시스템의 손쉽게 시스템을 확장함으로써 대용량 데이터 분석에 용이하다. 또한 데이터는 분산 자장되고 복사본을 유지하여 노드 데이터를 안전하게 보호할 수 있다. 또한 Discovery 기능이 내장되어 별도의 분산 시스템 관리자가 필요하지 않다.

○ Multi-tenacy 지원: 하나의 데이터를 여러 개의 분리된 인덱스들의 그룹으로 저장할 수 있다. 즉, 데이터의 속성에 따라 인덱스를 다르게 할당하여 유연한 검색이 가능하다.

○ 문서 (JSON) 포맷 구조 사용: JSON 구조로 인하여 모든 레벨의 필드에 쉽고 빠르게 접근이 가능하며 사전 매핑 없이 JSON 문서 형식으로 데이터를 바로 입력하여 색인 작업을 수행할 수 있다. 또한 REST API 지원을 통해 HTTP Method 행태로 사용이 가능하다.

○ 실시간 분석 지원: 저장된 데이터는 검색을 위해 별도의 갱신을 필요로 하지 않는다. 즉, 색인 작업이 완료됨과 동시에 바로 검색이 가능하다.

3. 구현 방안

클러스터 운영하는데 있어 다수의 시스템에서 발생하는 다양한 이벤트 로그 또는 상태 정보를 일관된 형태로 관리하는 기법은 매우 중요하다. 이를 위해 ELK 소프트웨어 스택을 이용하여 물리적인 장비에 장착된 센서로부터 측정한 정보(온도, 팬 속도, 소비전력), 시스템 자원 활용 현황 (CPU, 메모리, 디스크 사용량), 작업 수행 및 큐 현황 (작업 시작 시간, 작업 종료 시간, 사용된 자원량, 큐 상태)를 동일한 형태로 일원화 하여 관리하도록 한다.

○ 노드들에서 쏟아져나오는 로그(대표적으로 Syslog)는 Filebeat를 통해 수집/전송

○ 소위 ELK (ElasticSearch / Logstash / Kibana)를 통해 로그를 수집/가공/저장/검색 및 시각화

✓ 각 노드에서의 로그 수집 (Ship Logs) - Filebeat / Topbeat: ELK에서는 packetbeat, filebeat, topbeat, winlogbeat등 다양한 Beat를 통해 실시간으로 데이터를 수집한다. 본 보고서에서는 이중 filebeat와 topbeat에 대해서 알아본다. Filebeat는 syslog와 같이 /var/log에 쌓이는 로그 파일을 수집하여 전송한다. Topbeat는 이름에서 확인할 수 있듯이 리눅스 명령어의 Top과 유사하게 아래의 시스템 정보를 실시간으로 수집하여 elasticsearch로 전송한다.

☞ system: 시스템 자원 상태 수집. (system load, cpu useage, memory useage, swap useage)

☞ process: 프로세스별 분석데이터 수집. (process name, parent pid, state, pid, cpu useage, memory useage)

☞ filesystem: 파일시스템 분석데이터 수집. (마운트된 디스크의 총량, 사용량, 각 디스크의 이름, 마운트위치 등)

☞ cpu_per_core: core별 cpu 사용량 수집

✓ 로그의 처리와 인덱싱 (Process & Filtering) - Logstash: Filebeat, Topbeat를 비롯하여 다양한 로그 데이터를 가공하여 출력한다. 이를 위해 Input, Filter, Output으로 구성된 pipeline 구조로 구성된다.

☞ Input: 수집할 데이터를 정의한다. 기본적으로 약 40종 이상을 제공하며 Apache accesslog나 log4j에서 발생하는 로그를 수집하기 위해서는 일반적으로 file이나 tcp를 이용하여 데이터를 수집한다.

☞ Filter: input으로 정의된 로그에 대해서 parsing을 지원하기 위한 설정이다. filter에서 사용되는 RegExp 패턴들은 logstash 폴더 내 patterns 폴더 하위에 적재되어 있는데 필요한 경우 이 pattern들을 활용하여 데이터를 좀 더 쉽게 가공할 수 있다.

☞ Output: 가공 처리된 로그 데이터를 elasticsearch로 전송한다.

✓ 로그의 저장(Store & indexing) - ElasticSearch: 수집된 로그를 저장하고 검색을 지원하기 위해 인덱스를 생성하여 관리한다. elasticsearch는 기본적으로 검색엔진이지만 NoSQL처럼 비정형 데이터베이스로 사용이 가능하다. 데이터 모델을 JSON으로 사용하고 있어서, 요청과 응답을 모두 JSON으로 주고 받고 소스 저장도 JSON 형태로 저장한다.

✓ 로그의 검색과 시각화 (Visualize Logs) - Kibana: 저장된 로그 데이터를 기반으로 가시화한다. 즉 사용자가 검색 질의 (query)를 보내면 이를 elasticsearch에 전달하고 검색 결과를 화면에 가시화해주는 자바 기반 웹 어플리케이션이다.

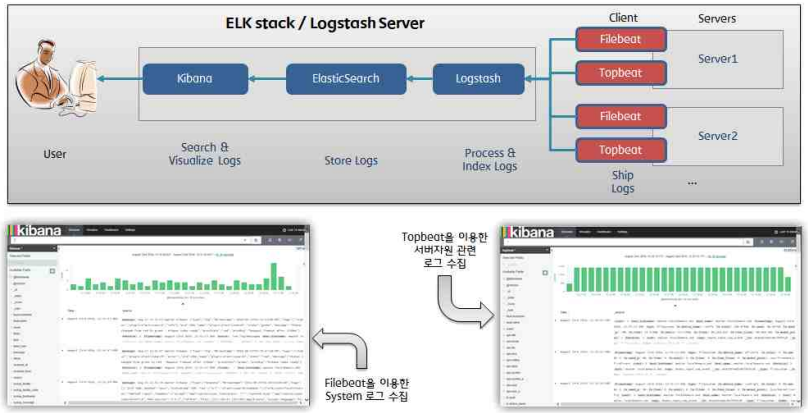


그림 10. ELK를 통한 로그 수집 및 분석 프레임워크.

제 3 장 슈퍼컴퓨터 운영 관리 시스템

제 1 절 아키텍처

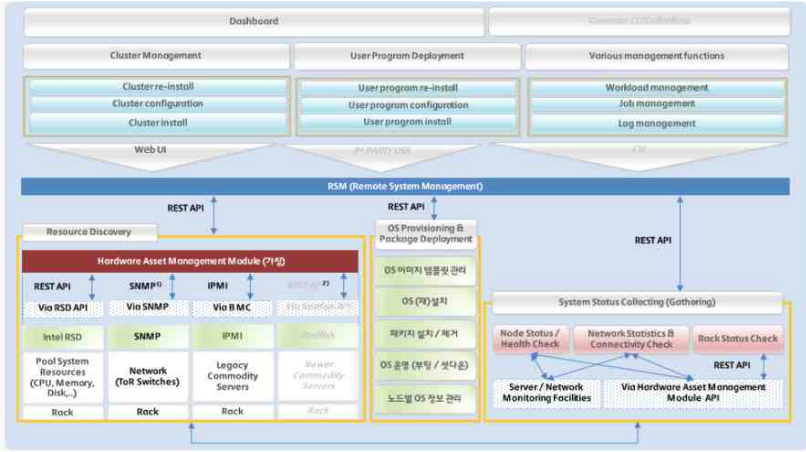


그림 11. 차세대 슈퍼컴퓨터 운영관리 시스템 아키텍처.

<그림 11>은 차세대 슈퍼컴퓨터 운영관리 시스템의 아키텍처를 보여준다. 구현된 운영관리 시스템은 관리 대상 시스템(Intel RSD, OCP 서버, Regacy 서버, ToR 스위치)에 기능을 플러그인 형태로 시스템에 적용하여 노드들마다 각기 다를 수 있는 시스템 관리 인터페이스의 지원을 위해 기능 모듈화 (IPMI / Redfish / SNMP / Intel RSA)로 구현하였다.

○ 벤더들마다 다른 하드웨어와 관리 도구에 별도로 액세스를 하지 않고 일원화된 API를 이용함.

○ 신규 하드웨어에서 새로운 시스템 관리 인터페이스 형식이 등장할 경우 이를 모듈 형태로 개발하여 지원 가능함.

○ 과거 도입된 HPC 하드웨어의 시스템 관리 인터페이스도 모듈 형태로 개발하여 지원할 수 있어 기존 하드웨어도 새로운 운영 시스템에 통합하는 것이 용이해짐.

제 2 절 구현 결과

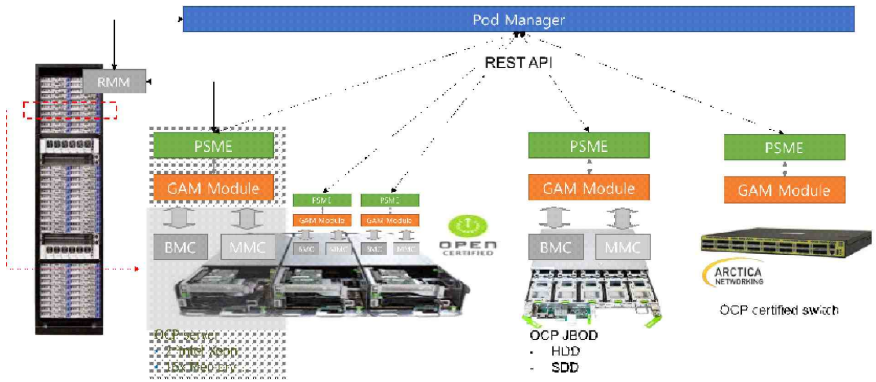


그림 12. OCP 장비를 통한 Intel RSD 테스트베드 구축 설계도.

<그림 12>는 OCP 장비를 이용하여 Intel RSD 테스트 베드를 구축한 설계도이다. 본 그림에서 확인할 수 있듯이 OCP 서버에 PSME 모듈을 설치하여 PoD Manager를 통해 서버 정보를 수집하고 수집된 결과를 구현된 대시보드를 통해 사용자에게 보여준다.

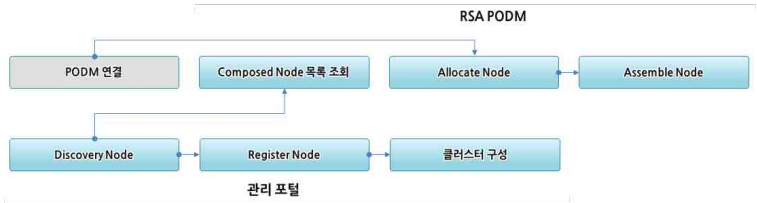


그림 13. Intel RSD POD Manager API 연동.

□ Intel RSD PODM API 가 제공하는 노드 정의 명령("/redfish/v1/Nodes/Actions/Allocate")을 이용하여 노드를 정의하고 노드 생성 명령("/redfish/v1/Nodes/1/Actions/ComposedNode.Assemble")을 호출하면, Logical Node가 생성됨.

□ 관리포털은 PODM의 정의된 Node 목록("/redfish/v1/Nodes")을 주기적으로 모니터링하여 신규 생성된 노드를 발견함.


```
{
  "@odata.context": "/redfish/v1/$metadata#Nodes",
  "@odata.id": "/redfish/v1/Nodes",
  "@odata.type": "#ComposedNodeCollection.ComposedNodeCollection",
  "Name": "Composed Nodes Collection",
  "Members@odata.count": 5,
  "Members": [
    {
      "@odata.id": "/redfish/v1/Nodes/1"
    },
    {
      "@odata.id": "/redfish/v1/Nodes/2"
    },
    {
      "@odata.id": "/redfish/v1/Nodes/3"
    },
    {
      "@odata.id": "/redfish/v1/Nodes/4"
    },
    {
      "@odata.id": "/redfish/v1/Nodes/5"
    }
  ],
  "Actions": {
    "#ComposedNodeCollection.Allocate": {
      "target": "/redfish/v1/Nodes/Actions/Allocate"
    }
  }
}
```


Composed Logical Node

Node 생성 API

그림 14. Intel RSD 연동 화면.

□ 발견한 노드정보를 관리자에게 알람으로 전달하고, 관리자 확인 후 등록자산으로 관리함.

□ 등록된 RSD Logical Node는 이 후 일반 서버와 동일하게 인식됨으로써 클러스터에 할당할 수 있음.

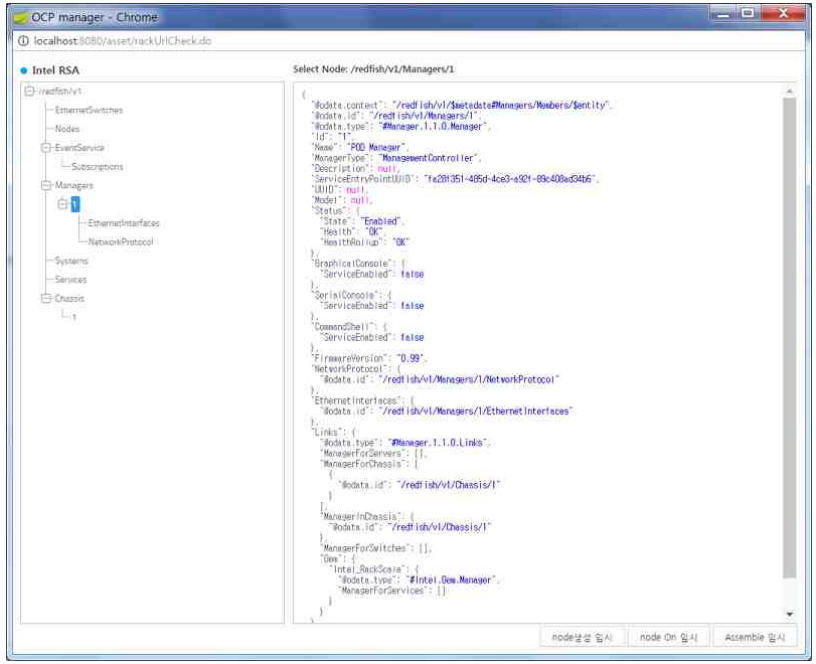


그림 15. Intel RSD 모니터링 구현 화면.

제 3 절 측정 지표 및 제공 정보

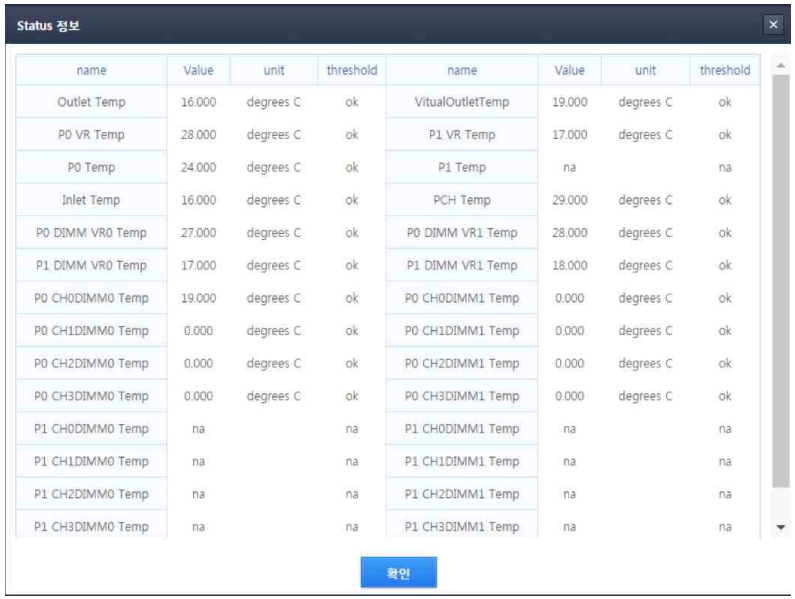
<그림 11>에서 보는 바와 같이 차세대 슈퍼컴퓨터 운영관리 시스템에서는 자원의 형태에 따라 관리 모듈이 개별적으로 존재하며 측정 지표 및 제공 정보는 다음과 같다.

○ 컴퓨팅 자원 관리 모듈

✓ IPMI를 통해 BMC로부터 다음 정보를 수집하여 제공한다.

☞ 컴퓨팅 보드 온도

☞ 컴퓨팅 보드 팬 속도



Status 정보

name	Value	unit	threshold	name	Value	unit	threshold
Outlet Temp	16.000	degrees C	ok	VitualOutletTemp	19.000	degrees C	ok
P0 VR Temp	28.000	degrees C	ok	P1 VR Temp	17.000	degrees C	ok
P0 Temp	24.000	degrees C	ok	P1 Temp	na		na
Inlet Temp	16.000	degrees C	ok	PCH Temp	29.000	degrees C	ok
P0 DIMM VR0 Temp	27.000	degrees C	ok	P0 DIMM VR1 Temp	28.000	degrees C	ok
P1 DIMM VR0 Temp	17.000	degrees C	ok	P1 DIMM VR1 Temp	18.000	degrees C	ok
P0 CH0DIMM0 Temp	19.000	degrees C	ok	P0 CH0DIMM1 Temp	0.000	degrees C	ok
P0 CH1DIMM0 Temp	0.000	degrees C	ok	P0 CH1DIMM1 Temp	0.000	degrees C	ok
P0 CH2DIMM0 Temp	0.000	degrees C	ok	P0 CH2DIMM1 Temp	0.000	degrees C	ok
P0 CH3DIMM0 Temp	0.000	degrees C	ok	P0 CH3DIMM1 Temp	0.000	degrees C	ok
P1 CH0DIMM0 Temp	na		na	P1 CH0DIMM1 Temp	na		na
P1 CH1DIMM0 Temp	na		na	P1 CH1DIMM1 Temp	na		na
P1 CH2DIMM0 Temp	na		na	P1 CH2DIMM1 Temp	na		na
P1 CH3DIMM0 Temp	na		na	P1 CH3DIMM1 Temp	na		na

확인

그림 16. BMC 수집 정보 구현 화면.

○ 네트워킹 자원 관리 모듈

✓ <그림 17>에서 보는 바와 같이 구현된 슈퍼컴퓨터 운영관리 시스템에서는 SNMP를 통해 ToR 스위치의 MIB 정보를 수집하여 제공한다. 이를 통해 아래의 정보를 확인할 수 있다.

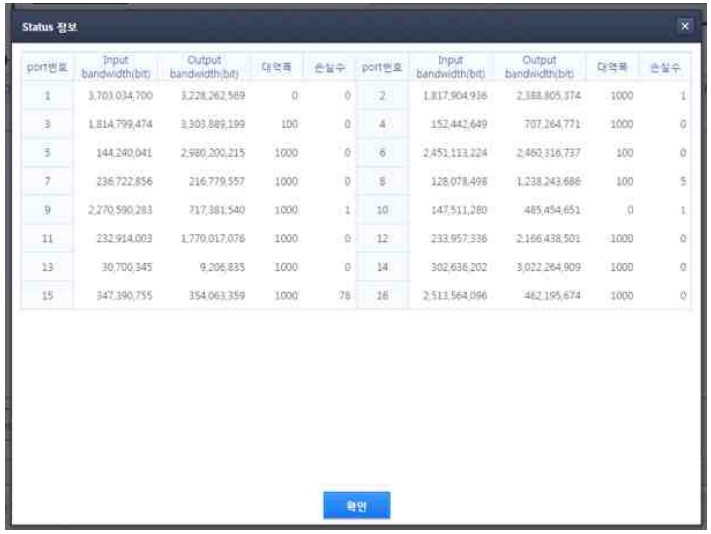
☞ 포트별 동작 상태

그림 17. 스위치 포트 상태 정보 구현 화면.

☞ 포트별 가용 대역폭

☞ 포트별 입력/출력 대역폭

☞ 포트별 패킷 손실률



port번호	Input bandwidth(bit)	Output bandwidth(bit)	대역폭	손실수	port번호	Input bandwidth(bit)	Output bandwidth(bit)	대역폭	손실수
1	3,703,034,700	3,228,262,569	0	0	2	1,817,904,936	2,388,805,374	1000	1
3	1,814,799,474	3,303,889,199	100	0	4	152,442,649	707,264,771	1000	0
5	144,240,041	2,980,200,215	1000	0	6	2,451,113,224	2,462,316,737	100	0
7	236,722,856	216,779,557	1000	0	8	128,078,498	1,238,243,686	100	5
9	2,270,590,283	717,381,540	1000	1	10	147,511,280	485,454,651	0	1
11	232,914,003	1,770,017,076	1000	0	12	233,957,336	2,166,438,501	1000	0
13	30,700,345	9,206,835	1000	0	14	302,636,202	3,022,264,909	1000	0
15	347,390,755	354,063,359	1000	78	16	2,513,564,096	462,195,674	1000	0

그림 18. 스위치 포트별 상세 정보 구현 화면.

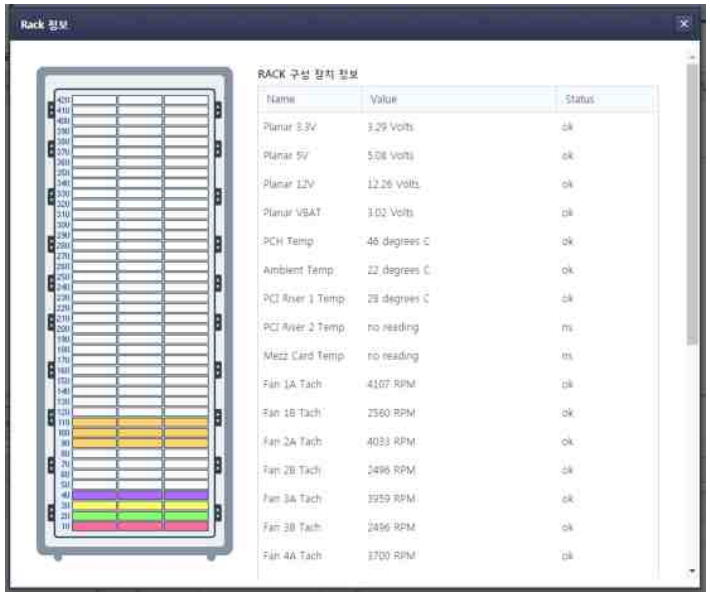
○ 인프라 자원 관리 모듈

✓ OCP 랙의 경우 RACK으로 입력되는 AC 전원 (3P4W 180 ~ 264 VAC 50A)을 바 형태의 12V DC 전원으로 변환하여 서버에 제공한다. 또한 RMC (Rack Management Controller)를 통해 관리자는 원격으로 아래의 정보를 확인할 수 있으며 팬 ON/OFF 관리가 가능하다.

☞ 랙 전체의 소비 전력

☞ 랙의 팬 속도

☞ 서버 위치 정보



Name	Value	Status
Planar 3.3V	3.29 Volts	ok
Planar 5V	5.08 Volts	ok
Planar 12V	12.26 Volts	ok
Planar VBAT	3.02 Volts	ok
PCH Temp	46 degrees C	ok
Ambient Temp	22 degrees C	ok
PCI Riser 1 Temp	28 degrees C	ok
PCI Riser 2 Temp	no reading	ns
Mezz Card Temp	no reading	ns
Fan 1A Tach	4107 RPM	ok
Fan 1B Tach	2560 RPM	ok
Fan 2A Tach	4033 RPM	ok
Fan 2B Tach	2496 RPM	ok
Fan 3A Tach	3959 RPM	ok
Fan 3B Tach	2496 RPM	ok
Fan 4A Tach	3700 RPM	ok

그림 19. 랙 상세 정보 구현 화면.

○ 작업 상태 관리 모듈

✓ 스케줄러인 SLURM의 로그를 통해 아래의 정보를 수집하여 제공한다.

☞ 스케줄러에 의해 관리되는 실행 노드 정보

☞ 실행 노드 파티션 정보

☞ 작업 ID

☞ 작업을 제출한 사용자

☞ 작업 상태

☞ 작업 시간

☞ 작업이 할당된 노드

squeue data

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
20	normal	PI_Test_01	test001	CG	05:21	2	node[001-002]
21	normal	PI_Test_02	test001	CG	03:34	1	node003
22	normal	PI_Test_03	test001	R	02:51	2	node[004-005]
23	normal	PI_Test_04	test001	PD	00:00	2	(Resources)
24	normal	PI_Test_05	test001	PD	00:00	3	(Resources)
25	normal	PI_Test_06	test001	PD	00:00	1	(Resources)
26	normal	PI_Test_07	test001	PD	00:00	4	(Resources)
27	normal	PI_Test_08	test001	PD	00:00	3	(Resources)

그림 20. 작업 상세 정보 출력 화면.

sinfo data

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
Normal	up	48:00	5	idel	node[001-005]

그림 21. 자원 상세 정보 출력 화면

제 4 장 머신러닝 기법을 통한 시스로그 분석 방안[1] 제언

제 1 절 필요성

일반적으로 시스템 관리자는 시스템에 결함이 발생한 경우 결함의 원인을 파악하기 위해 이벤트 로그 데이터를 분석한다. 이벤트 로그 파일에는 Warning, Error, Critical 등 결함의 등급을 의미하는 Severity level 값뿐만 아니라 결함의 원인을 추론할 수 있는 메시지를 포함하고 있기 때문에 시스템 관리자는 이벤트 로그의 메시지를 분석함으로써 결함의 발생 원인을 추론할 수 있다. 그러나 최근 들어 시스템의 규모가 커지고 복잡해짐에 따라 발생하는 이벤트 로그 데이터의 양이 많고 시스템 구성 모듈 간 관계도 복잡해 관리자가 이벤트 로그 파일을 하나하나 분석하여 결함 원인을 파악하는 것은 거의 불가능하다.

또한 현재의 시스로그 분석 시스템의 경우 대부분 정해진 필드를 기반으로 데이터를 수집한다.

그림 22. LogAnalyzer 출력 화면.

<그림22>은 시스로그 분석을 위한 오픈소스 프로그램인 LogAnalyzer[6]의 결과화면이다. 그림에서 확인할 수 있듯이 보여지는 필드는 RFC[7]에서 규정한 데이

1) 본 절은 다음 논문에서 중요 부분만 발췌함. 박주원, 김은혜, 염재근, 김성호, "시스템 결함 분석을 위한 이벤트 로그 연관성에 관한 연구," 한국산업경영시스템학회지 제39권 제2호, 129-137 페이지, 2016년 6월.

터 포맷과 일치한다.

Field	Descript
ID	Event log message ID
Received Time	The received time of the syslog event in server.
Device Reported Time	The reported time of the syslog event in device.
Facility	The facility code to specify the type of program that is logging the message.
Severity Level	The severity code to indicate relative importance.
Fromhost	The machine that originally sent the syslog event.
Message	A free-form message that provides information about the event.

표 3. Syslog 출력 형식.

즉, 대부분의 경우 시스로그 이벤트 메시지에 있는 데이터 필드를 발생 시간에 따라 나열하고 키워드를 통한 필터링 기능을 제공하는 경우가 대부분이다. 그러나 이 경우 다음과 같은 문제가 있다.

첫째, 중복된 로그 메시지가 많다. 대부분의 에러 메시지의 경우 문제가 해결되지 않은 동안 일정한 주기로 반복되어 로그 메시지가 출력된다. 예를 들어 NTP (Network Time Protocol) 서버에 연결상 문제가 발생한 경우 주기적으로 에러 메시지가 출력된다. 이 경우 운영자 입장에서는 시스템 전체 운영에 있어 치명적인 문제가 아니기 때문에 대부분 수정하지 않은 경우가 많아 동일한 메시지가 주기적으로 출력된다. 둘째, 에러의 원인 파악은 어렵다. 시스로그 메시지를 통해 원인을 파악하기 위해서는 대부분 관리자가 메시지 내용을 분석해야 한다. 그러나 <표3>과 같은 형식으로 메시지가 출력될 경우 관리자는 에러의 원인을 규명하기 위해서 모든 메시지 내용을 확인해야 한다. 셋째, 에러 메시지간의 연관성을 파악하기 어렵다. 일반적으로 하나의 근본적인 원인 (Root Cause)으로 인해 많은 형태의 에러 메시지가 발생할 수 있다. 예를 들어 Network Interface Card 드라이버 문제가 발생한 경우 NTP 서버, 스케줄러 서버등 네트워크를 통한 모든 연결상의 에러 메시지가 동시에 출력된다. 그러므로 이를 분석하기 위해서는 동일한 행태의 메시지를 그룹화하고 발생 연관성을 파악해야 근본적인 원인을 유추해 볼 수 있다.

제 2 절 제안 방법

본 절에서는 시스템에서 발생한 근본적인 결함을 검출하고 운영자로 하여금 에러 메시지의 가독성을 향상시키기 위해 메시지 내용을 기반으로 한 클러스터링 기법을 제시한다. 또한 최근 많이 사용하는 오픈소스 기반의 클라우드 관리 플랫폼 오픈 스택 시스템을 대상으로 로그 메시지를 수집하고 수집된 제시된 클러스터링 기법을 적용한 예를 살펴본다.

1. 로그 수집

오픈 스택의 경우 nova, neutron, keystone 등 다양한 서비스 모듈이 연동되어 실행되는 클라우드 관리 플랫폼으로 일반적으로 하나의 서버가 아닌 다수의 서버에 분산되어 실행된다. 또한 실행되는 서비스에 따라 서로 다른 파일에 로그가 출력된다. 예를 들어, 컴퓨팅 노드에 설치되는 NOVA 서비스의 경우/var/log/nova 파일에, NEUTRON의 경우 /var/log/neutron 파일에 이벤트 로그가 각각 출력된다.

이와 같이 다수의 서버에 다수의 파일로 혼재되어 있는 로그 파일을 하나의 서버로 수집하기 위해 시스로그데몬을 활용 할 수 있다. 즉, 오픈 스택 이벤트 로그 메시지를 <표>에서 제시한 포맷으로 시스템 로그에 출력하고 이를 rsyslog 클라이언트를 통해서 서버로 전송한다.

이를 위해 먼저, rsyslog 서버를 아래와 같은 설정을 통해 활성화하여 rsyslog 클라이언트로부터 이벤트 로그 데이터를 수집하고 이를 데이터 베이스에 저장한다.

```
$ModLoad immark
$ModeLoad imtcp
$InputTCPServerRun portNum
$ModLoad ommysql
 *.* :ommysql:database-server,database-name,database-userid,database-password
```

둘째, 오픈 스택에서 출력되는 메시지를 시스로그 형태로 시스템 로그 파일에 출력 되도록 오픈 스택의 각 서비스 설정 파일에 아래와 같은 설정을 추가 한다.

```
use_syslog=True
syslog_log_facility=LOG_LOCAL0
```

셋째, 시스템 로그에 출력된 로그 메시지를 수집하여 rsyslog 서버로 전송하기 위해 오픈 스택을 구성하는 모든 노드의 rsyslog.conf 파일에 아래의 설정을 추가

하여 rsyslog 클라이언트를 활성화한다.

2. 전처리

오픈스택은 분산된 다수의 서버에서 다양한 서비스 모듈이 실행되고 서로 연동되어 운영되기 때문에 이벤트 로그 메시지의 종류가 다양하고 수도 매우 많다. 그러므로 필터링을 통해 중복된 이벤트 로그 메시지를 제거하고 메시지 필드의 내용 중에서 중요한 메시지만을 추출하여 정규화하는 전처리 작업이 필요하다. [8]에서 제시한 바와 같이 추출된 메시지 내용을 의미론적으로 유사한 이벤트 로그로 군집화하기 위해 다음 정규화 과정을 실행한다.

먼저, 모든 문자를 소문자로 변환한다. 대소문자는 의미론적으로 구분이 없음에도 불구하고 대부분의 프로그래밍 언어에서 구분하여 처리하는 경우가 많다. 특히, 데이터 마이닝 분야에서 많이 활용되는 언어 중 하나인 R[9]의 경우 대소문자를 구분하여 비교하기 때문에 모든 문자를 소문자로 변환한 후 유사성을 계산한다. 둘째, 의미를 담고 있는 단어가 아닌 기호, 숫자, 조사 등을 제거한다. 예를 들어, -, +, ‘, : 와 같은 기호나 a, the, of, such와 같은 관사 및 조사를 제거함으로써 군집화의 정확도를 높일 수 있다. 셋째, ID와 같은 고유값을 제거한다. 오픈스택의 경우 모든 가상 인스턴스에 ID를 부여하여 관리한다. 그러므로 동일한 현상의 이벤트 로그 메시지에도 각각 다른 인스턴스 ID가 포함되어 있기 때문에 ID을 제거하여 정확도를 높인다. 넷째, 디렉토리 경로는 PATH라는 단어로 대체한다. 오픈스택의 로그에는 파일의 위치를 표기하는 디렉토리 경로가 다수 포함되어 있으나 의미론적 입장에서는 파일의 위치를 설명하는 부수적인 의미를 내포하기 때문에 디렉토리 경로를 PATH라는 단어로 변경하여 군집화를 진행한다.

3. 메시지 기반 군집화

전처리 작업을 통해 정규화된 메시지를 의미론적으로 유사한 것으로 군집화하기 위해 먼저 메시지 전체를 단어별로 분류하고 단어의 사용여부와 빈도수를 나타내는 메트릭으로 변환한다. 변환된 메트릭으로 유사도를 계산하기 위해 유클리디안 거리를 계산한다. 유클리디안 거리는 일반화된 유클리디안 공간에서의 거리를 의미하여 아래의 식으로 계산된다.

$$\langle u, v \rangle = \sum_{i=1}^{n} u_i v_i$$

그림 23. 유클리디안 거리

이렇게 계산된 유클리디안 거리를 기반으로 계층적 군집분석을 통해 유사도가 일정값(threshold)보다 큰 경우 하나의 클러스터로 구성함으로써 군집화한다.

제 3 절 실험 결과

1. 실험 환경

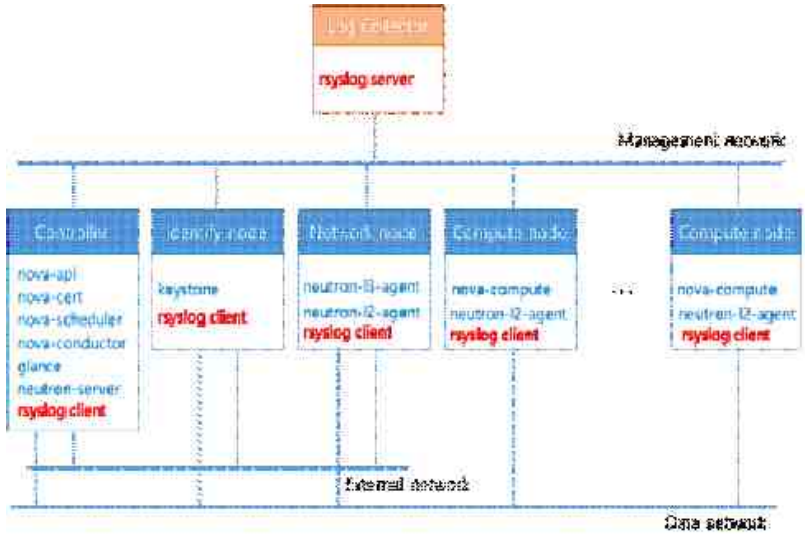


그림 24. 실험 환경.

<그림 24>은 실험에서 사용한 오픈 스택의 플랫폼에 이벤트 로그 수집을 위해 rsyslog를 설치한 그림이다. 그림에서 보는 바와 같이 실험에 활용된 오픈 스택 플랫폼은 총 31대로 구성되어 있으며 28대의 컴퓨팅노드와 네트워크노드, 제어노드, 인증노드로 1대씩을 사용하고 있다. 오픈스택 커뮤니티에서 제공하는 메뉴얼에서 제시한 바와 같이 각 서비스 모듈간 통신을 위한 관리 네트워크와 실제 데이터 전송을 위한 데이터 네트워크를 별도로 구성하였으며 외부 연결을 위해 제어노드, 인증노드, 네트워크노드만 외부 네트워크에 연결하였다. 시스로그데몬에서 출력되는 이벤트 로그 데이터와 오픈스택의 로그 데이터를 서버로 전송하기 위해 모든 오픈

스택 노드에 rsyslog 클라이언트를 설치하고 관리 네트워크에 rsyslog 서버를 연결하여 이벤트 로그 데이터를 수집하였다.

2. 실험 결과

FromHost	OpenStack Module	Facility	# of log	class
Controller	root	16	724	6
Controller	neutron.common.legacy	16	1	7
Network node	neutron.common.legacy	16	2	7
Network node	neutron.agent.l3_agent	16	3	8
Network node	neutron.agent.l3_agent	16	5	9
Login node	N/A	3	360	2
Compute node 1 ~ 28	nova.virt.libvirt.driver	16	32206	1
Compute node 3	N/A	0	65	3
Compute node 3	N/A	0	65	4
Compute node 3	N/A	0	65	5

표 4. 군집화 결과.

<Table 6>는 수집된 이벤트 로그의 메시지를 기반으로 의미론적으로 군집화한 결과를 보여준다. <Table 6>의 첫 번째 필드는 이벤트 로그 데이터가 발생한 호스트 정보이다. 두 번째 필드는 메시지가 발생한 오픈스택 모듈을 나타내는 필드이다. 오픈 스택에서 발생하는 이벤트 로그 메시지에는 로그가 발생한 서비스 모듈이 표기되기 때문에 결함이 발생한 모듈을 쉽게 파악할 수 있다. 세 번째 필드는 Facility code로 오픈스택의 경우 16(LOG_LOCAL0) 값으로 수집되었음을 확인할 수 있다. 네 번째 필드는 발생한 이벤트 로그 수이며 다섯 번째 필드는 메시지 기반으로 군집화한 결과이다.

본 실험에서는 오픈 스택의 모듈 필드와 Facility 필드 값을 이용하지 않고 메시지에 담겨있는 텍스트만을 이용한 비지도 학습 방식의 군집화 기법을 사용하였다. 그럼에도 실험 결과에서 확인할 수 있듯이 두 번째, 세 번째 필드와의 연관성을 확인할 수 있다. 즉, 오픈스택에서 발생한 이벤트 로그의 경우 neutron.agent.l3_agent 모듈에서 발생하는 이벤트 로그만 8번과 9번 그룹으로 구분되었을 뿐 나머지 모듈에서 발생한 이벤트 로그 데이터는 하나의 그룹으로 구분되어 있음을 확인할 수 있다.

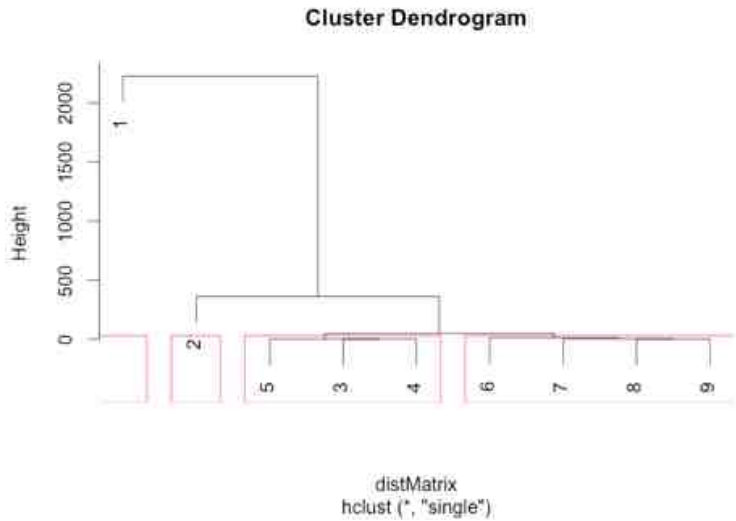


그림 25. 유클리디안 거리 기반 군집화 결과 화면.

<그림 25> 그룹화된 메시지 간의 유클리디안 거리를 기반으로 군집화한 결과를 보여준다. 그림에서 확인할 수 있듯이 그룹 3.4.5번과 그룹 6.7.8.9는 연관성이 있음을 확인할 수 있다. 실제로 그룹 3,4,5번의 경우 모두 3번 노드에서 발생한 이벤트 로그 메시지로 SATA HDD 연결 문제로 인하여 발생하는 커널 메시지로 확인되었다. 또한 그룹 6,7,8,9의 경우에는 제어노드의 Neutron-server와 네트워크 노드의 Neutron l3 agent에서 발생하는 이벤트 로그 메시지로 두 모듈간 연결에 결함이 있어서 발생한 이벤트 로그 메시지이다.

참고 문헌

(1) 권원옥, et. al. “마이크로서버 기술 동향,” Electronics and Telecommunications Trends, vol. 29, no. 4 2014. pp. 49-58.

(2) B.B. Lowekamp. "Combining active and passive network measurements to build scalable monitoring systems on the grid." Performance Evaluation Review, 30(4): 19-26, 2003.

(3) J.-W. Park and J.Kim, "Quality monitoring for real- time IP media transport over multi-point delivery environment," IEEE Transaction on Consumer Electronics, 54(4):2060-2067, Nov. 2008.

(4) R. Gerhards. The syslog protocol. RFC 5424. 2009.

(5) A. B. Downey and D. G. Feitelson, “The elusive goal of workload characterization,”ACM SIGMETRICS Performance Evaluation Review, vol. 26, pp. 14-29, 1999.

(6) Reips, Ulf-Dietrich, and Stefan Stieger. "Scientific LogAnalyzer: A Web-based tool for analyses of server log files in psychological research." Behavior Research Methods, Instruments, & Computers 36.2 (2004): 304-311.

(7) Gerhards, R., The syslog protocol, RFC 5424, 2009.

(8) Pitakrat, T., Grunert, J., Kabierschke, O., Keller, F., and Hoorn, A., “A framework for system event classification and prediction by means of machine learning,” in Proc. of the 8th International Conference on Performance Evaluation Methodologies and Tools. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014, pp. 173-180.

(9) R development core team, R : A language and environ- ment for statistical computing, [Online], Available : http://www.r-project.org.

※ 저자 정보

이름	소속/연락처
박주원	슈퍼컴퓨터시스템개발실/juwon.park@kisti.re.kr
조혜영	슈퍼컴퓨터시스템개발실/chohy@kisti.re.kr
박주연	솔리드ENG/cypark@solideng.co.kr