워크플로우 모델링을 통한 IaaS 클라우드 플랫폼 구축 최적화

Optimized Construction of an IaaS Cloud Platform with Workflow Modeling

박달용*, 최상훈*, 최동훈**, 박기웅* 1)
Dal-Yong Park, Sang Hoon Choi, Dong Hoon Choi, and Ki-Woong Park

(300-716) 대전시 동구 대학로 62 대전대학교 해킹보안학과 시스템보안연구실*
(305-701) 대전시 유성구 대학로 245 한국과학기술정보연구원 정보소프트웨어센터**
pdy0709@naver.com, csh0052@gmail.com, choid@kisti.re.kr, woongbak@dju.kr

요 약

IaaS(Infrastructure as a Service) 클라우드 시스템인 OpenStack은 복수개의 소프트웨어 구성요소로 이뤄지며 각 소프트웨어 컴포넌트들의 구축에 따른 시간적 오버헤드가 크고 약 100여 가지의 구축 명령어를 통해 구축되므로, 시스템을 구축함에 상당한 비효율성이 따른다. 본 논문에서는 기존의 OpenStack 기반 클라우드 시스템 구축 절차를 워크플로우 관점으로 프로파일링 하여, 구축자 관점의 OpenStack 설치 워크플로우 도면 및 워크플로우 기반 의존성 판별 알고리즘을 통해 최적화된 OpenStack 기반 클라우드 플랫폼 설치 Script를 도출하기 위한 구축 프레임웍을 제안하였다. 기존 OpenStack 공식문서에서 제시된 방법과 달리, 각 소프트웨어 컴포넌트별 의존성과 연관성을 워크플로우 관점으로 프로파일링하여 개선된 OpenStack 설치 Script는 최대 약 23%까지 구축속도 향상을 보인다.

Abstract

IaaS(Infrastructure as a Service) cloud system such as OpenStack consist of several software components. Constructing the OpenStack apts to be a time-consuming task because the OpenStack construction requires about one hundred command for downloading, deploying, and configuring each software component. The task leads to user-obstructive latency. As a remedy to these problems, we propose an optimized script for constructing OpenStack-based cloud system by workflow-based dependency detection algorithm. The experimental results of our scheme show that the total elapsed time to complete the OpenStack deployment is up to 23% more time-efficient in comparison with the OpenStack construction procedure.

키워드: 클라우드 컴퓨팅, 오픈스택, 워크플로우, 모델링
Keyword: Cloud Computing, OpenStack, Workflow, Modeling

1) 교신저자

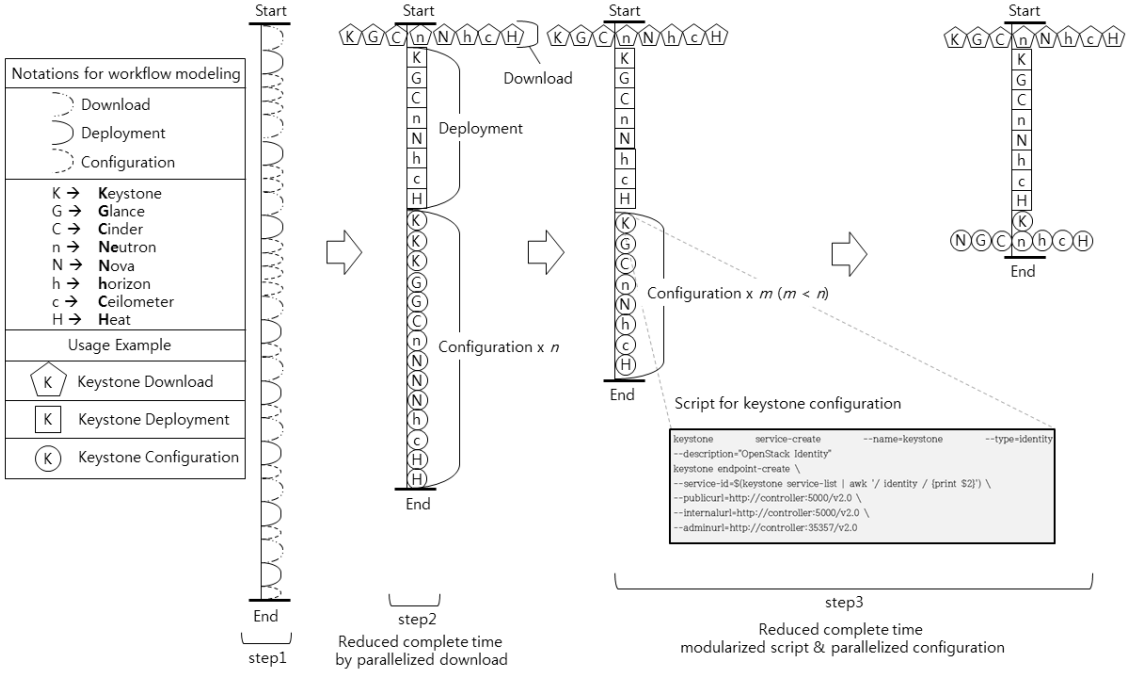
논문투고일자 : 2014. 11. 26 심사일자 : 2014. 12. 01 게재확정일자 : 2014. 12. 22

1. 서론

최근 클라우드 시장이 성장하고 기관별 대규모 클라우드 플랫폼을 구축하는 IT 전략에 따라 클라우드의 효율적인 구축은 주요 이슈가 되고 있다[1]. IaaS 클라우드 플랫폼 구축을 위한 오픈소스 기반 소프트웨어는 OpenStack[2], CloudStack[3], Eucalyptus[4], Opennebula[5] 등 다양하게 존재하지만, IaaS 클라우드 시스템 중 하나인 OpenStack은 가장 널리 사용되는 오픈소스 기반 클라우드 시스템 구축 소프트웨어로써, 주어진 물리적 서버 자원에서 사용자가 원하는 가상머신을 생성하여 제공하는 가상화 기술 및 자원 관리 및 모니터링을 위한 기술을 지원한다. 하지만 IaaS 클라우드 시스템을 구축하는 데는 상당한 시간적/연산적 비용이 소요된다. 예를 들어, OpenStack 공식문서[6]에 따른 클라우드 플랫폼 구축을 수행할 시 약 30여 가지의 환경설정 파일 설정 및 약 100가지의 설치 명령어를 실행해야 하므로, 구축에 따른 시간적 오버헤드가 소요된다. 이와 같은 문제점을 해결하기 위하여, OpenStack 기반 클라우드 플랫폼 구축을 용이하게 만들기 위한 DevStack[7]이 개발되었다.

DevStack은 OpenStack의 스크립트를 이용하여 모든 소프트웨어 컴포넌트를 일괄적으로 구축하는 소프트웨어 툴이다. DevStack은 일괄적인 설치를 위하여 미리 정의된 스크립트를 통해 기존 방법에 비해 구축을 용이하게 하였으나, 여전히 순차적인 다운로드와 구축으로 인한 시간적 오버헤드 발생과 설치되는 시스템 환경에 따른 환경설정 시 오류가 발생하여 구축에 어려움이 따른다[8].

따라서 본 논문에서는 이와 같은 문제를 해결하기 위하여, 기존 OpenStack 기반 클라우드 플랫폼 구축을 위한 각 절차를 워크플로우 관점[9]으로 분석을 위한 1단계, 도출된 알고리즘을 토대로 제작된 워크플로우[10]를 통해 프로파일링을 수행하는 2단계, 분석 및 프로파일링 결과를 기반으로 최적화된 설치 방법을 도출하는 3단계로 구성된 전략을 기반으로 연구를 수행하였다. 워크플로우(Workflow)

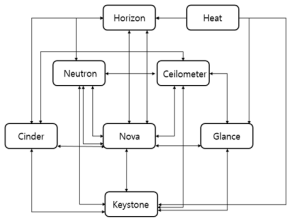


(그림 1) 효율적인 OpenStack 구축 방안을 위한 단계별 전략

란 하나의 업무 수행하는데 필요한 세부 프로세스를 객체로 정의하고, 각 객체별 수행 관계를 정의하여 절차를 분석하기 위한 모델링 기법이다[11]. 본 논문에서는 OpenStack 기반 클라우드 시스템을 구축하기 위한 프로세스에 워크플로우 모델링 기법을 적용하여 구축 절차를 프로파일링하고 워크플로우 관점에서의 분석을 진행하였다. 이를 통해 구축자 관점의 OpenStack 설치 워크플로우 도면 및 워크플로우 기반 의존성 판별 알고리즘을 통해 최적화된 OpenStack 기반 클라우드 플랫폼 설치 Script를 도출하였다[12].

(그림 1)에서 도시 된 바와 같이 OpenStack 문서에 기반 해 각 구축 절차에 따른 도면제작 및 프로파일링을 수행하기 위한 1단계, 분석 및 프로파일링 결과를 기반으로 워크플로우 모델링[14]을 수행하고, 최적화된 구축절차를 도출하기 위한 2단계, 분석 및 프로파일링 결과를 기반으로 Script를 제작하고, Script들을 효율적으로 설치하는 Main script를 추출하는 3단계로 구성된 전략으로 연구를 수행하였다[13].

본 논문의 구성은 다음과 같다. 2장에서는 OpenStack 기반 클라우드 시스템을 구축하기 위한 소프트웨어 컴포넌트의 다운로드/설치 및 설정 의존성을 파악하기 위한 OpenStack 설치 도면을 제시함과 동시에 프로파일링을 토대로 알고리즘[15]을 제시하고 구축 워크플로우를 제작한다. 3장은 2장의 분석 결과를 토대로 Script 기반의 효율적인 구축방안을 제시한다. 4장에서는 개선된 해결된 해결책에 따른 속도향상을 위한 실험 결과를 나타내었으며, 5장에서는 본 논문의 결론을 제시한다.



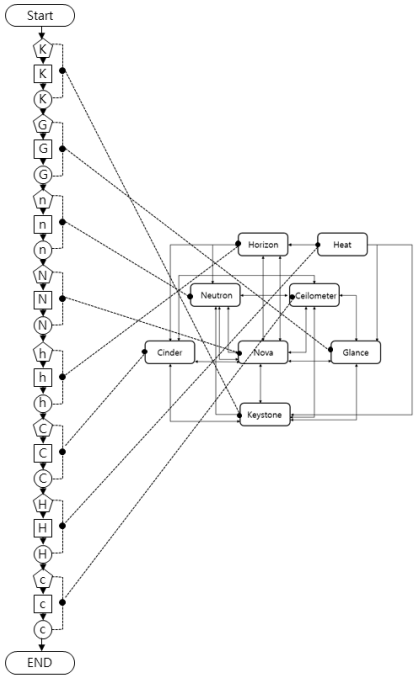
소프트웨어 컴포넌트	특징
Keystone(Identity)	모든 소프트웨어 컴포넌트들의 인증관리
Glance(Image)	OS의 구동을 위한 가상머신들의 이미지 관리
Cinder(Block Storage)	블록 스토리지를 관리하여 하드디스크 역할의 컴포넌트
Neutron(Network)	OpenStack의 모든 네트워크 관리
Nova(Compute)	가상머신의 생성,관리를 위한 컴포넌트
Horizon(Dashboard)	오픈스택 구현 서비스 제어를 위한 웹 인터페이스
Ceilometer(Telemetry)	OpenStack의 벤치마킹, 확장성, 및 통계의 목적을 위한 모니터링과 미터링을 지원
Heat(Orchestration)	OpenStack을 위한 서비스들의 통합, 편성을 지원

(그림 2) OpenStack의 각 소프트웨어 컴포넌트별 기능

2. 워크플로우 기반 OpenStack 구축 프로파일링

2.1 OpenStack 구축 절차에 대한 도면 제작 및 프로파일링

본 장은 1단계 연구수행에 해당하는 장으로서 효율적인 OpenStack 기반 IaaS 클라우드 시스템 구축을 위한 프로파일링을 수행한다. OpenStack은 주어진 물리적 서버 자원에서 사용자가 원하는 가상머신 및 가상 스토리지/네트워크를 생성하여 제공하며, 자원 가상화 기술과 함께 자원 관리 및 모니



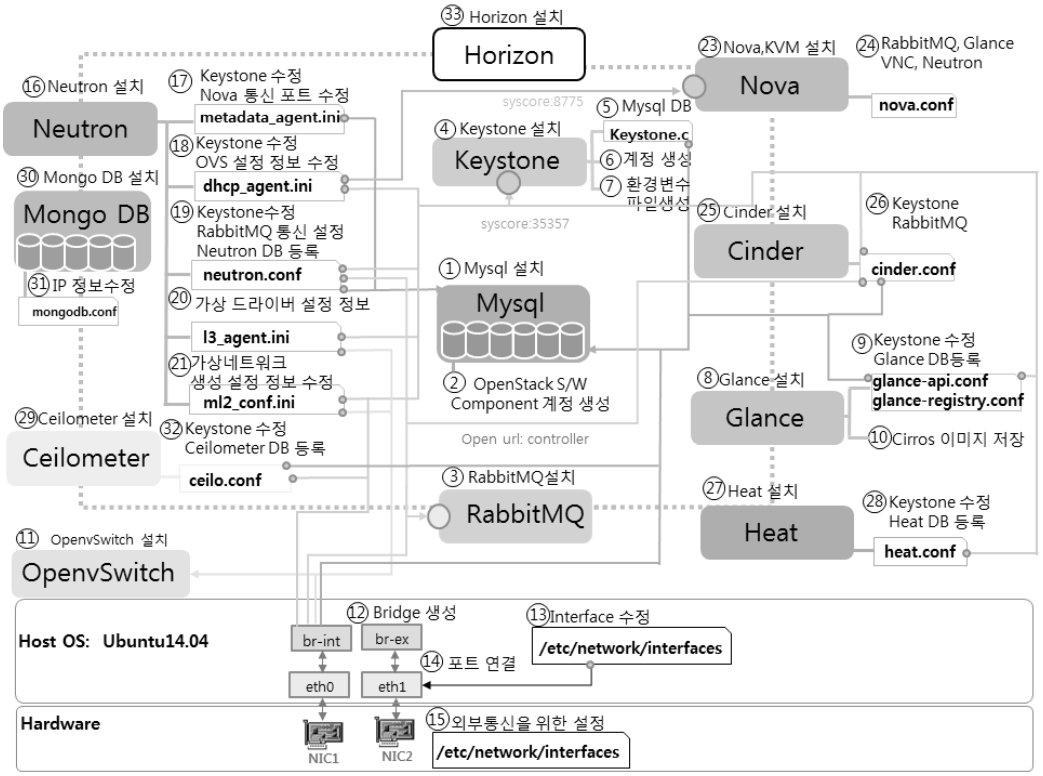
(그림 3) OpenStack 기반 IaaS 구축 절차

터링을 위한 기술을 지원한다. 이러한 기능을 수행하기 위해 OpenStack은 총 8개의 소프트웨어 컴포넌트들로 구성되어 있으며 각 역할은 (그림 2)에 도시된 바와 같다. OpenStack은 내부를 구성하는 8개의 내부 소프트웨어 컴포넌트들이 모두 설치 및 환경설정이 정상적으로 완료된 후 동작하게 된다. OpenStack에서 제공하는 공식문서는 OpenStack을 구성하기 위한 소프트웨어 컴포넌트 별로 순차적인 다운로드, 설치/설정 및 구성을 위한 방법을 제시한다. 하지만 기존 OpenStack 구축 가이드라인에 따른 구축에는 다음과 같은 두 가지의 문제점이 존재한다.

첫째, 설치자는 각 구축명령어가 OpenStack의 전체 설치과정 및 환경설정 시 어떤 소프트웨어 컴포넌트에 의존성 및 영향이 있는지 파악하기 힘들다. 따라서 OpenStack 공식문서에서 제시하는 하드웨어 구성과 상이한 시스템에 구축을 할 경우 가상 네트워크 자원 할당문제, 소프트웨어 컴포넌트 간 연동 및 내부 통신 환경설정 오류 등으로 많은 어려움이 따른다[17]. 둘째, (그림 3)과 같이 OpenStack 구축 가이드라인에 따라 구축이 진행될 경우, 각 설치 단계 중 상당수가 독립적으로 설치가 가능하여, 각 소프트웨어 컴포넌트들의 다운로드, 설치 및 환경설정이 병렬적으로 진행 될 수 있음에도, 순차적으로 진행되어 비효율적 구축이 이루어진다.

따라서 본 논문에서는 효율적인 구축을 위하여, 다음과 같은 두 가지의 해결 방안을 제시한다.

첫째, (그림 4)에 도시된 바와 같이, OpenStack 구축 문서를 통해 구축자 관점에서의 각 구축단계별 관계도 및 의존성을 용이하게 파악하기 위한 구축 관계도를 제작하였다. 관계도는 각 구축 단계 별 소프트웨어 컴포넌트 간의 연동을 위한 관계 및 환경설정에 따른 의존성을 나타낸다. 또한 제작된 관



(그림 4) OpenStack 문서 기반 구축 관계도

계도는 네트워크 연결을 도식화 하여 각 소프트웨어 컴포넌트 별 상호작용을 위한 port와 url 뿐만 아니라 Customized 환경설정과 가상네트워크를 나타내어 소프트웨어 컴포넌트별 오류 시 오류를 찾아내는데 용이하도록 하였다.

둘째, OpenStack의 효율적인 구축을 위한 워크플로우 최적화 알고리즘을 제안하였다. 본 논문에서 제안하는 워크플로우 최적화 알고리즘은 OpenStack 내부 소프트웨어 컴포넌트의 다운로드, 설치 및 환경설정 간의 의존성을 파악하고, 최적화된 설치 단계를 도출하기 위한 알고리즘이다. 제안한 알고리즘은 2.2장에 기술하였다.

2.2 프로파일링 알고리즘 디자인

2단계에 해당하는 본 절은 OpenStack 소프트웨어 컴포넌트들의 다운로드, 설치, 설정의 의존성 판단을 통해 최적화된 설치 단계를 워크플로우 관점으로 도출하기 위한 알고리즘을 제안한다.

제안하는 <알고리즘 1>은 워크플로우로 모델링된 구축과정의 최적화를 위한 알고리즘이다. 알고리즘의 입력은 T, C 이며 출력은 DR, IR 이다. 입력 값인 T는 소프트웨어 컴포넌트의 집합으로 정의하며, 집합 T는 복수개의 t로 구성된다. 여기서 t란 task의 약자로서, 각 소프트웨어 컴포넌트를 OpenStack에 탑재시키기 위한 작업으로 정의되며 t_n으로 표기된다(라인1).

C는 각 t_n별 종속된 집합이며, 복수개의 c로 구성된다. 여기서 c란 command의 약자로서, 각 소프트웨어 컴포넌트 별, 다운로드, 설치, 설정 명령어로 정의된다(라인2).

출력에 해당되는 DR은 의존성을 갖는 task쌍의 집합이며, IR은 의존성이 없는 task쌍의 집합이다. 예를 들어, DR의 요소인 task쌍($\{t_i, t_j\}$)은 선행 task인 t_i가 수행된 이후 다음 task인 t_j가 수행되어야 정상적인 구축으로 이어질 수 있는 task쌍을 의미한다. 하지만 의존성이 나타나지 않는 IR의 요소인 task쌍($\{t_i, t_j\}$)은 t_i 및 t_j의 실행순서와 무

〈알고리즘 1〉 OpenStack의 효율적인 구축을 위한 워크플로우 최적화 알고리즘

```
1  Input: T = {t_0,.....,t_m}                          //task
2         C = {_0c_0,.....,_mc_n}                      //command
3  Ouput: DR = {{t_1,t_1}.....{t_i,t_j}}
4         IR = {{t_1,t_1}.....{t_i,t_j}}
5  Begin
6    While _mP_2 Times (m = # of Tasks ∈ T)
7      t_i,t_j ← Select two Tasks ∈ T
8      While _nP_2 Times (n = # of (Commands ∈ {t_i,t_j})
9          S1 ← Command Seq. # ∈ t_i
10         S2 ← Command Seq. # ∈ t_j
11         If (Dependency check (_ic_s1 , _jc_s2) == true)
12             Dependency_flag = true
13         End If
14     End While
16      If (Dependency_flag == true)
17         DR ← (t_i,t_j)
18      Else
19         IR ← (t_i,t_j)
15      End If
20   End While
21 End
```

〈알고리즘 2〉 의존성 파악 알고리즘

```
1 Input: _ic_s1 , _jc_s2
2 Output: true, false
3   Dependency check()
4    Begin
5      Dependency check between _ic_s1 , _jc_s2
6       IF {_ic_s1 ∩ _ic_s2 }=∅
7          output true
8       Else
9          output false
10      End IF
11 End
```

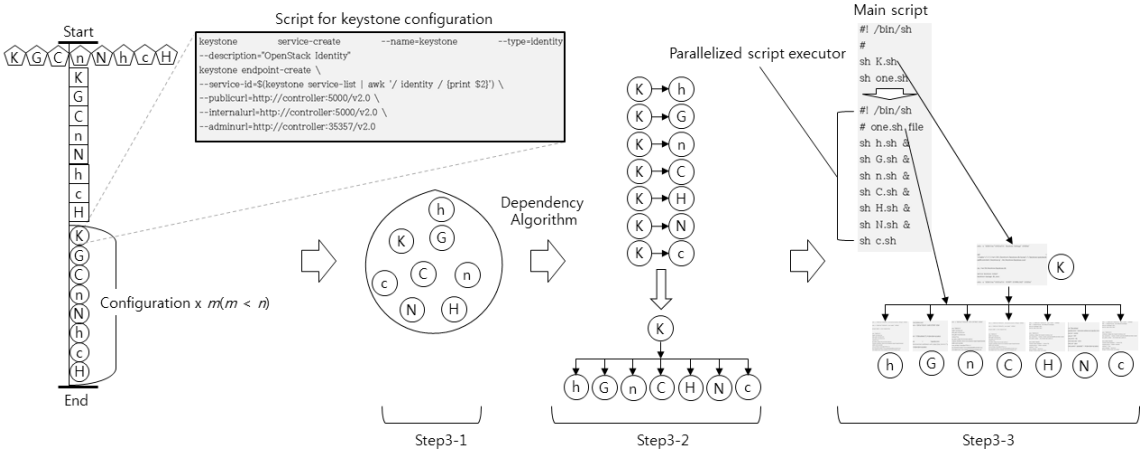
관하게 정상적인 구축으로 이어질 수 있는 Task쌍을 의미한다. 예를 들어 IR 집합의 요소에 task쌍 $\{t_i, t_j\}$이 존재하고 t_i는 Nova, t_j는 Cinder의 다운로드를 위한 task로 정의될 경우, Nova와

Cinder의 각 다운로드 task는 서로 의존성이 존재하지 않아 병렬적으로 수행될 수 있다는 것을 의미한다. DR 집합의 요소에 task쌍$\{t_i, t_j\}$이 존재하고 t_i는 Keystone, t_j는 Glance의 환경설정을 위한 task로 정의될 경우, Keystone과 Glance의 환경설정 연산은 서로 의존성이 존재하여, t_i를 수행 후 t_j를 수행해야 오류가 발생하지 않는다는 것을 의미한다. <알고리즘 1>은 모든 task쌍에 대한 의존성을 판단하기 위하여 반복적으로 수행한다. 반복횟수는 task쌍의 모든 조합의 개수인 ${}_mP_2$ 및 각 task쌍 내부 명령어의 모든 조합의 개수인 ${}_nP_2$으로 정의되어 총 ${}_mP_2 \times {}_nP_2$ 번 수행된다(라인6-8). 각 반복 연산에 있어서는 명령어 쌍$({}_ic_{s1}, {}_jc_{s2})$에 대한 의존성 판단을 수행하며 이는 <알고리즘 2>의 Dependency check 함수를 사용하여 각 명령어간 종속성을 판단하여 의존성을 검출한다. <알고리즘 2>는 명령어 쌍$({}_ic_{s1}, {}_jc_{s2})$를 입력으로 받아, 각 명령어간 종속성이 존재할 경우 true를 반환하며, 존재하지 않을 경우 false를 반환한다. 각 task쌍별 반복 연산에 있어, Dependency_flag가 true일 경우 해당 task쌍은 의존성이 있다는 것을 의미하므로, DR에 저장되며, false일 경우 해당 task쌍은 서로 독립적으로 수행될 수 있다는 것을 의미하므로, IR에 저장된다(라인11~15). 반복 연산이 끝난 후에는 DR과 IR에는 종속성 여부가 판별된 task쌍이 분류되고 알고리즘의 수행이 종료된다.

3. OpenStack 구축 최적화

3단계에 해당하는 본 장에서는 2.2절에서 디자인한 알고리즘을 이용하여 종속성을 추출하여 효율적인 구축 방안을 제시한다.

먼저, 각 소프트웨어 컴포넌트들의 다운로드, 설치, 설정 종속성 여부 판단을 위해 2.2절에서 도출된 알고리즘을 수행한다. 그 결과, 다운로드 task들은 모든 task들이 IR 집합에 저장되었으며, 설치 task들은 모두 DR 집합에 저장되었다. 설정 task쌍은 IR={{K,h} {K,G} {K,n} {K,C} {K,H} {K,N} {K,c}}와 같은 결과가 도출되었다. (그림 5)의 step3-1에 도시화된 바와 같이, 순차적으로 실행되던 소프트웨어 컴포넌트들의 설정을 알고리즘을 통한 종속성 확인을 토대로 최적화한다. 결과를 토대로 우선순위가 먼저인 Keystone 설정이 완료된 뒤, 나머지 소프트웨어 컴포넌트들의 설정이 병렬적 수행되는 워크플로우로 (그림 5)의 step3-2와 같이 나타낼 수 있다. 구축 단계 워크플로우가 병렬적으로 이루어짐에 따라 시간적 오버헤드 절감이 이루어지며, 구축 최적화가 이루어진다. 본 논문에서는 Script를



(그림 5) 알고리즘을 통해 도출된 의존성과 Main Script를 통한 최적화 구축 방안

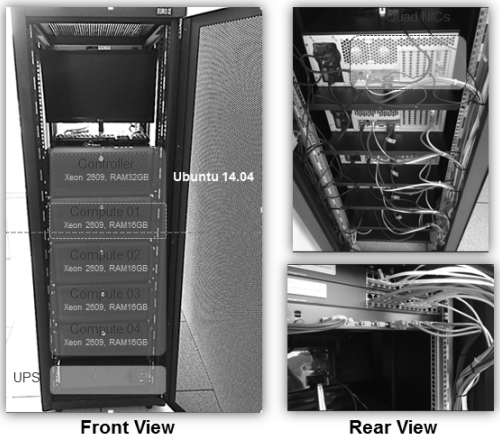
통해 병렬적인 구축을 진행하기 위해 Keystone 설정이 완료된 상태에서 나머지 설정들이 병렬적으로 구축되도록 Main Script를 제작하였다. (그림 5)의 step 3-3에 도시된 바와 같이 Main Script는 Keystone 서비스 컴포넌트가 수행이 모두 끝난 뒤 나머지 모든 컴포넌트들이 병렬적 구축이 이루어질 수 있도록 동작하여 시간적 오버헤드를 절감한다.

4. 성능 평가

본 절에서는 본 논문에서 제안한 최적화 방법에 따른 구축시간 향상을 측정하기 위한 실험 환경 및 분석결과를 나타낸다.

4.1 실험환경

실험환경은 (그림 6)과 같이 5대로 구성된 서버는 CPU Xeon E5-2609(2.5GHz)를 사용하였고 5대중 1대는 Memory 32GB를 사용하였으며 4대는 Memory 16GB를 사용한다. Storage 환경은 SSD로 진행하였다. 실험에 사용된 OpenStack version은 13년 10월에 출시된 Hanava와, 최근 14년 04월에 출시된 Icehouse를 최적화 시켜 각 OpenStack version 별 구축시간 향상을 측정하기 위한 실험환경을 구축하였다.



(그림 6) OpenStack 물리적 구축 환경

4.2 분석 결과

본 장에서는 OpenStack 공식문서를 통한 구축 방안과 본 논문에서 제시한 워크플로우기반의 구축 방안의 구축 시간비교를 통해 OpenStack 구축 시간 향상도를 분석한다.

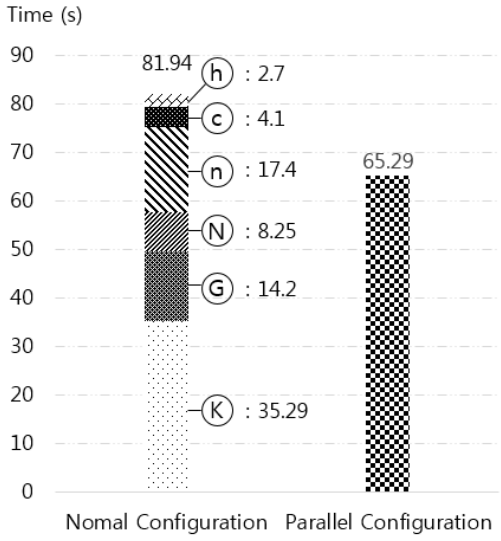
첫 번째 실험은 다운로드 및 설치 시간을 측정하기 위한 것으로서, OpenStack 공식문서에서 제공하는 순차적인 다운로드 및 실행 방안과 알고리즘을 통해 도출된 병렬적인 다운로드 및 설치 방안에 따른 속도 개선 정도를 측정한 결과를 나타낸다. OpenStack의 Havana version과 Icehouse version을 이용해 총 두 가지 version에서 다운로드 및 설치 시간 비교를 진행하였다. OpenStack 공식문서에서 제공하는 방안을 통한 다운로드 및 설치를 진행하였을 때 <표 1>과 같이 Havana version과 Icehouse version에서 각각 335/352초가 소요됨을 확인하였다. 알고리즘을 통해 도출된 병렬적 다운로드 및 설치를 진행하였을 때에는 각각 279/278초가 소요되어 알고리즘을 통해 도출된 병렬적 다운로드 및 설치로 약 23%의 속도 개선이 이루어짐을 확인하였다.

〈표 1〉 일반 다운로드 설치와 병렬 다운로드 설치 시간 비교

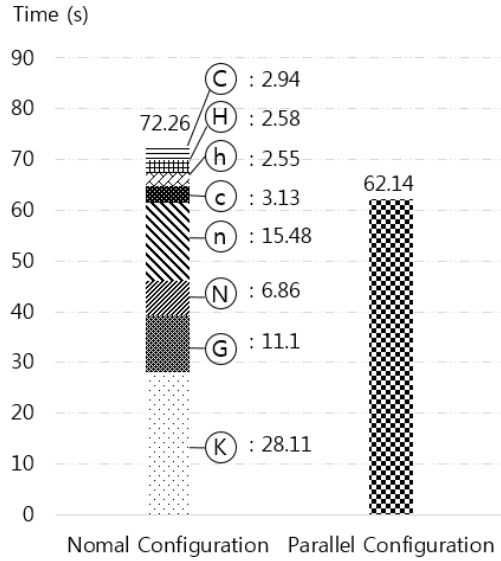
OpenStack Version	Havana	Icehouse
Download & Installation	335s	352s
Parallel Download & Installation	279s	278s

두 번째 실험은 스크립트를 통해 환경설정을 진행하였을 때의 설정 시간을 측정하기 위한 것으로서, 공식문서에서 제공하는 순차적인 설정 방안과 알고리즘으로 도출된 워크플로우와 Main Script를 통해 도출된 병렬적인 설정 방안에 따른 속도 개선 정도를 측정한 결과를 나타낸다. OpenStack의 Havana version과 Icehouse version을 이용해 총

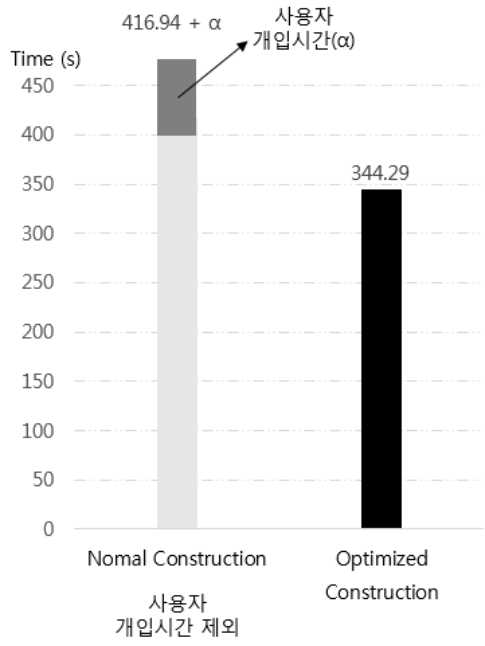
두 가지 version에서 환경설정을 수행하였다. (그림 7, 8)에 도식화 된 그래프와 같이 최적화하기 전 환경설정 시 81.94/72.76초가 소요되며 워크플로우와 Main Script를 통한 병렬적인 환경설정 진행 시 65.29/62.14초가 소요되어 약 21.25%의 시간 단축이 이루어짐을 확인하였다.



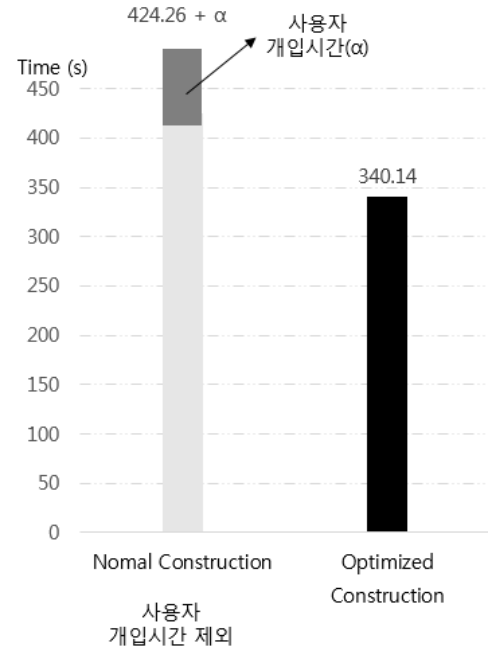
(그림 7) Hanava Configuration Time



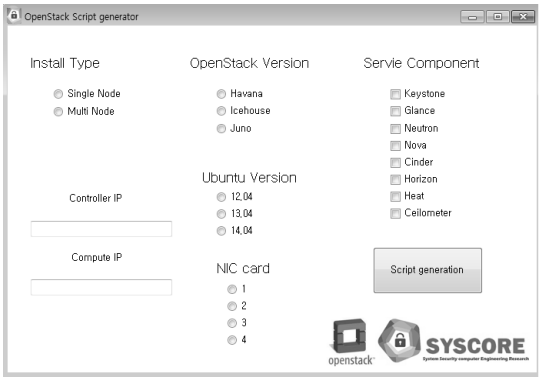
(그림 8) Icehouse Configuration Time



(그림 9) Havana Total Construction Time



(그림 10) Icehouse Total Construction Time



(그림 11) Script Generator (진행 연구)

세 번째 실험은 최적화하기 전 전체 구축 시간과 최적화 후 전체 구축 시간을 측정한 것이다. OpenStack의 Havana version과 Icehouse version에서 구축을 진행하였으며, (그림 9, 10)은 최적화하기 전 구축 소요 시간과 최적화 후 구축 소요시간을 비교하여 도식화 한 것이다. OpenStack의 구축 소요 시간 측정 시 개선 전 소요시간은 약 416.94/424.26초의 시간이 소요된다. 개선 후 설치 소요시간은 약 344.29/340.14초로 최대 23%의 속도 개선이 이루어짐을 확인하였다. 위 실험 결과 중 Normal Construction 부분은 사용자 개입시간을 제외하고 측정한 결과값으로서, 개입시간을 고려할 경우 성능 향상은 훨씬 더 높게 이루어 질 것이다.

5. 결론

본 논문에서는 일반적인 IaaS 클라우드 시스템인 OpenStack 구축과정에서 도면 제작과 이를 기반으로 한 일반화된 알고리즘을 제작과 워크플로우 모델링을 통해 각각의 다운로드 설치 및 소프트웨어 컴포넌트 의존성과 연관성을 분석하였으며, 이에 따라 개선된 구축 방안을 제시하였으며 성능 평가를 통하여 개선 전후의 시간을 측정하여 개선된 설치방안을 통해 공식 구축 방안보다 최대 23%의 속도 개선을 나타내어 클라우드 시스템을 구축하는데 시간적 오버헤드를 크게 줄일 수 있으며, 대규모 클라우드 시스템 구축 시 구축시간을 대폭 앞당길 수 있다. 또한 사용자 개입시간이 추가될 때 훨씬 더 높은 성능 향상이 도출될 것이다. 본 논문의 결과는 OpenStack에 적용하여 최적화된 결과를 도출했지만, 본 논문에서 제시된 워크플로우 모델링 기반 최적화 기법은 Hadoop과 같은 빅데이터 플랫폼에 적용하여 구축 최적화에 활용될 수 있을 것이다. 추후 연구로서 설치과정 및 구축 환경에 따른 Parameter를 사용자로부터 입력받아, 각 환경에 따른 최적화된 Script를 생성하여 구축할 수 있도록 Script Generator를 제작할 예정이다.

참고문헌

〔1〕 클라우드 지원센터 "2014년 상반기 클라우드 시장/정책 동향 보고서", 2014.06

〔2〕 OpenStack, http://www.openstack.org, 2014

〔3〕 CloudStack, http://cloudstack.apache.org/, 2014

〔4〕 Eucalyptus, http://www.eucalyptus.com/eucalyptus-cloud /iaas, 2014

〔5〕 OpenNebula, http://opennebula.org/, 2014

〔6〕 OpenStack Guide, http://docs.openstack.org

〔7〕 DevStack, devstack.org, 2014

〔8〕 Devstack Intall guide, http://docs.openstack.org/developer/ devstack/stack.sh.html, 2014

〔9〕 이승욱, "유전자 알고리즘을 활용한 효율적인 워크플로우 업무처리에 관한 연구", 한국경영과학회, 2008.11

〔10〕 박성주, 김현아, 김광훈, "워크플로우 소셜 네트워크 근접구심도 분석 알고리즘", 한국인터넷정보학회, 2012.11

〔11〕 안현, 김현아, 김광훈, "워크플로우 기반 업무-수행자 대응 분석 기법", 한국인터넷정보학회, 2012

〔12〕 최정열, "클라우드 데이터 센터의 에너지 효율성 평가 프레임워크", 한국차세대컴퓨팅학회, 2014

〔13〕 이준우, "클라우드 컴퓨팅 기반의 대용량 이동객체 분산 처리 시스템", 한국차세대컴퓨팅학회, 2012.02

〔14〕 황경순, 이건명, "적합성과 선호도를 고려한 워크플로우의 적응적 자원 스케줄링", 한국퍼지 및 지능 시스템 학회, 2004
〔15〕 Haibo Li, Dechen Zhan, "Workflow Timed Critical Path Optimization", Nature and Science, 2005
〔16〕CloudeAVE, http://www.cloudave.com/15100/deploying-openstack-is-difficult-its-a-myth/, 2011

저자소개

◆ 박달용

- 2014년: 대전대학교 전산정보보호학과 졸업(학사)
- 현재: 대전대학교 전산정보보호학과 재학중(석사과정)
- 관심분야: 클라우드보안, 시스템보안

◆ 최상훈

- 2014년: 대전대학교 전산정보보호학과 졸업(학사)
- 현재: 대전대학교 전산정보보호학과 재학중(석사과정)
- 관심분야: 클라우드보안, 시스템보안

◆ 최동훈

- 1983년: 한국과학기술원 전산학과 졸업(석사)
- 1989년: Northwestern University 전산학과 졸업(박사)
- 1992년~1999년: 동덕여자대학교 부교수
- 2005년~현재: 한국과학기술정보연구원 책임연구원
- 관심분야: 데이터베이스, 바이오 인포매틱스

◆ 박기웅

- 2005년 연세대학교 Computer Science 학사
- 2007년 KAIST Electrical Engineering 석사
- 2012년 KAIST Electrical Engineering 박사
- 2008년 Microsoft Research Asia, Wireless and Networking Grourp, Researcher
- 2009년 Microsoft Research Redmond, Network Research Group, Researcher
- 2012년 국가보안기술연구소 연구원
- 2012년~현재 대전대학교 해킹보안학과 조교수
- 관심분야: 시스템 보안, 모바일-클라우드 컴퓨팅, 보안 프로토콜, 디지털 포렌식 등