ISBN

GLOVE 데이터 적재 성능 향상기법 설계 및 구현

한국과학기술정보연구원

저자소개

이세훈 (Sehoon Lee)

Korea Institute of Science and Technology Information
Division of Supercomputing
Senior Researcher
Email : sehooi@kisti.re.kr

차 례

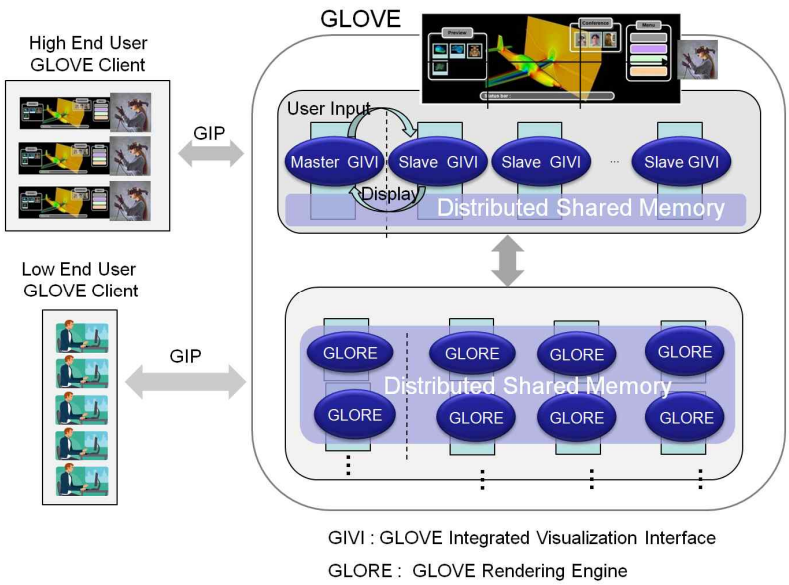
1. 개요

1.1. GLOVE(GLobal Virtual reality visualization Environment for scientific simulation)

GLOVE[1]는 고성능 컴퓨팅 환경에서 대용량 시뮬레이션 데이터를 효과적으로 분석하고 가시화하여 공유한다. 고해상도의 Tiled Display와 VR(Virtual Reality)환경은 응용 연구자에게 데이터 분석에 있어 직관적으로 다른 시각의 관점을 제공할 뿐만 아니라 다계층의 데이터 구조는 응용이 달라지더라도 최소한의 수정으로 응용에 맞는 하나의 새로운 인터페이스를 제공한다.

[그림 1]은 GLOVE 시스템의 구조를 보여준다. GLOVE는 사용자 인터페이스인 GIVI(GLOVE Integrated Visualization Interface)와 데이터 가공 및 렌더링을 담당하는 GLORE(GLOVE Rendering Engine)의 두 부분으로 이루어진다. GIVI는 가상현실 입출력 장치를 총괄하며, 사용자로부터 입력을 받아서 GLORE에 전달하고, GLORE의 실행 결과를 전달 받아서 화면에 출력한다. GLORE는 대용량 데이터의 가공 및 렌더링을 위한 GIVI의 서브 시스템이다.

GLOVE에서 GLORE와 GIVI는 서로 독립적으로 운영되는 클러스터에서 실행된다. 이러한 구조는 GLOVE의 인터페이스(GIVI)와 핵심 렌더링 기능(GLORE)을 물리적으로 분리할 수 있기 때문에 충분한 그래픽 처리능력을 갖추지 못한 시스템을 가지고 있는 사용자가 자신이 보유하고 있는 시스템에서 GIVI를 실행하고, 원격 장소의 고성능 렌더링 엔진인 GLORE를 이용해 대용량 데이터를 렌더링 할 수 있다.



[그림 1] GLOVE 시스템 구조

1.2. 대용량 데이터 가시화를 위한 분산공유메모리 기반 데이터 관리

대용량 데이터를 처리하고 만들어내는 계산과학 응용에 있어 데이터의 처리 방법은 매우 중요한 이슈이다. 때로 데이터들은 지리적으로 떨어진 곳에 분산되어 위치하기도 하며, 지리적으로 떨어져 있진 않더라도 그 양의 방대함으로 인해 데이터 저장소와 계산 처리가 다른 클러스터에서 이루어지기도 한다. 기후 모델링과 예측, 의학 이미지들, 고에너지 물리, 항공우주 시뮬레이션 등 오프라인으로 처리하기에도 힘든 테라(tera), 페타(peta) 바이트 단위의 데이터를 이들 모델링 도구들은 인터렉티브(interactive)하게 처리하여야 한다. 이러한 data-intensive한 응용들에 있어 성능의 병목은 데이터 스토리지와 원격 데이터 접근을 위한 지연이다. NASA의 기후 모델링, 분석, 예측(MAP)을 위한 프로젝트의 경우 실행 시간의 25~50%가 입출력을 기다리는 idle 시간이었다[2]. 실시간 data-intensive 응용들은 데이터의 접근과 사용에 대한 새로운 관점의 변화를 요구하였다.

- ▢ 여러 스토리지 시스템에 대한 병렬 접근
- ▢ 테라, 페타 바이트 수준의 데이터 처리
- ▢ 병렬 처리 시스템 간의 데이터 공유

대용량 데이터 가시화 또한 데이터의 입출력 패턴이 이들 계산과학의 응용들과 다르지 않다. 이에 더하여, 가시화는 대부분의 경우 계산과학이 부분적으로 만들어낸 데이터를 한 scene 내에 표현하여야 한다. 이러한 일을 수행하기에 한 시스템 내의 메모리는 턱없이 부족하다. 또한, 대용량 데이터의 가시화는 데이터 입출력 뿐 아니라 이미 메모리에 읽어 들인 데이터로 하나의 scene을 렌더링 하는 데에 걸리는 시간 지연도 또 하나의 병목이다. 이를 위해 여러 대의 클러스터가 렌더링을 나누어 수행하는 병렬 렌더링 기법을 사용한다. 병렬 렌더링 시, 데이터는 어떻게 나누어 공유할 것인지는 대용량 데이터 가시화의 또 다른 이슈이다.

대용량 데이터 가시화 도구 GLOVE는 대용량 데이터를 가시화하기 위한 효율적 데이터 관리 기술로 분산공유메모리 기반의 데이터 관리자를 구현하였다. 분산공유메모리는 HPC 클러스터의 메모리를 하나의 메모리처럼 사용하여 데이터를 적재할 수 있게 함으로써 한 번에 메모리에서 처리할 수 있는 데이터의 양을 클러스터 전체 메모리의 용량만큼 메모리에 적재할 수 있다. 그러나 이러한 기술은 가시화의 실시간 성능은 보장해 줄 수 있지만 가시화 기능을 수행하기 전 디스크에서 메모리로 데이터를 적재하는 시간이 매우 오래 걸린다는 단점이 있다. 이러한 단점을 보완하기 위해 본 문서에서는 데이터 적재 성능 향상을 위한 방안을 제시하고자 한다.

2. GLOVE 데이터 적재 성능 병목 분석

2.1. GLOVE 데이터 적재의 단계

분산병렬 데이터 가시화를 위해서 데이터 관리자는 원본 데이터 I/O, 메타 데이터 계산, 메타 데이터 분산공유메모리 영역 반영의 세 단계를 거쳐 디스크에서 메모리에 있는 분산공유메모리 영역으로 데이터를 적재한다. 본 절에서는 각 단계별 수행 과정을 설명하고, 전체 적재 과정에서 어느 단계에서 병목이 발생하는지 분석한다.

2.2. 원본 데이터 I/O

유동 해석 결과 데이터를 가시화하기 위해서는 데이터를 디스크에서 메모리로 적재해야 한다. GLOVE는 분산병렬처리를 위해 분산공유메모리를 사용하므로 적재를 수행하는 전체 컴퓨팅 노드에서 병렬 파일 시스템을 이용해 데이터를 각자의 분산공유메모리 영역에 적재한다. 다양한 형식의 데이터 파일 포맷을 지원하지만 변환을 거쳐 최종적으로 원하는 포맷은 효율적 병렬 입출력을 구현하기 위해 자체 개발한 GLOVE 데이터 포맷(이하 GLV 포맷)을 읽어 메모리에 적재하는 과정을 거친다. GLV 포맷은 데이터 셋에 대한 메타 정보를 meta.xml 파일에 관리한다. 메타 정보는 격자의 정렬/비정렬 여부, 타임스텝 정보, 격자가 시간에 따라 변하는 격자인지에 대한 정보 및 데이터 셋의 공통적인 특징을 포함한다. GLOVE 데이터 변환기는 CGNS, EnSight, OpenFOAM과 같은 비정렬격자 데이터 파일에서 필요한 메타 정보를 추출해 meta.xml 파일을 구성할 수 있어야 한다.

GLV 포맷은 효율적인 병렬 입출력을 구현하기 위해 설계된 KISTI의 가시화 도구 GLOVE에서 사용하는 데이터 포맷이다. GLV 포맷은 데이터 셋에 대한 메타 정보를 meta.xml 파일에 관리한다. 메타 정보는 격자의 정렬/비정렬 여부, 타임스텝 정보, 격자가 시간에 따라 변하는 격자인지에 대한 정보 및 데이터 셋의 공통적인 특징을 포함한다. GLOVE 데이터 변환기는 CGNS, EnSight, OpenFOAM과 같은 비정렬격자 데이터 파일에서 필요한 메타 정보를 추출해 meta.xml 파일을 구성할 수 있어야 한다. [표 1]은 비정렬격자 GLV 포맷의 meta.xml 파일의 내용 구성을 보여준다.

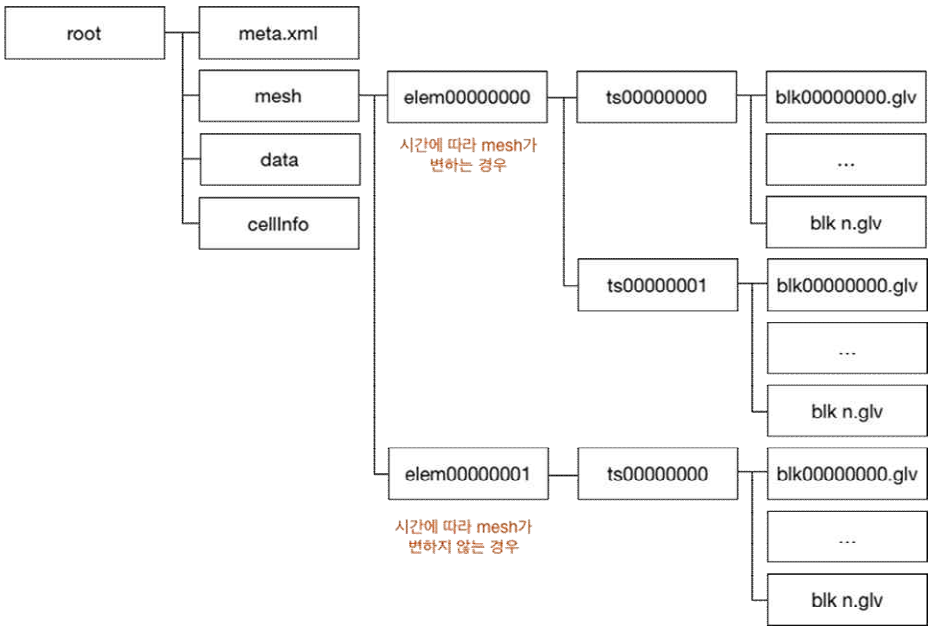
```
〈gdmMetaData〉
   〈gridType〉 USG 〈/gridType〉
   〈timestep〉 전체 타임스텝의 수 〈/timestep〉

   〈values〉
      〈value〉
         〈id〉 변수 ID 〈/id〉
         〈name〉 이름 〈/name〉
         〈vectors〉 변수를 구성하는 벡터 이름 (콤마로 구분) 〈/vectors〉
         〈type〉 데이터 타입 ( int | float | double ) 〈/type〉
      〈/value〉
      〈value〉
      ...
      〈/value〉
   〈/values〉

   〈elements〉
      〈element〉
         〈id〉 Element ID 〈/id〉
         〈name〉 이름 〈/name〉
         〈mesh〉
            〈dynamic〉 시간에 따른 격자의 변화 여부 ( true | false ) 〈/dynamic〉
            〈dimension〉 격자 차원 ( 1 | 2 | 3 ) 〈/dimension〉
            〈type〉 격자 데이터 타입 ( int | float | double ) 〈/type〉
         〈/mesh〉
         〈blockCount〉 Element가 포함하는 전체 블록 수 〈/blockCount〉
         〈values〉 Element가 포함하는 전체 변수 목록 (콤마로 구분) 〈/values〉
      〈/element〉
      〈element〉
      ...
      〈/element〉
   〈/elements〉
〈/gdmMetaData〉
```

[표 1] 비정렬격자 GLV 포맷을 위한 meta.xml 파일 구성

GLOVE는 데이터 적재(load)와 병렬 처리를 위해 유동 해석 솔버(solver)가 만든 데이터를 변환하여 다음과 같은 디렉토리 구조로 저장한다. GLOVE 데이터 디렉토리는 데이터 셋의 메타 정보를 기술한 meta.xml 파일과 3개의 서브 디렉토리(mesh, data, cellInfo)를 가진다.

Mesh 디렉토리는 격자 데이터를 저장한다. 격자는 실험자가 구성한 물리적 단위인 Element로 구분되고, 각 Element는 시변환(time-varying) 데이터를 고려하여 다시 타임스텝 별로 구분된다. Element-타임스텝 별로 분할된 격자 데이터 블록을 저장한다. 격자는 실험의 종류에 따라 시간에 따라 변하기도, 그렇지 않기도 한다. 그리고 Element 별로 시간에 따라 격자가 변하는지 여부를 설정할 수 있다. 격자가 일정한 경우(static mesh)는 타임스텝 0만 유지하고, 그렇지 않은 경우(dynamic mesh)는 각 타임스텝마다 별도로 격자를 저장한다. 최하위 디렉토리에 있는 blkN.glv 파일은 실제 격자 데이터를 저장한 바이너리(binary) 파일이다.

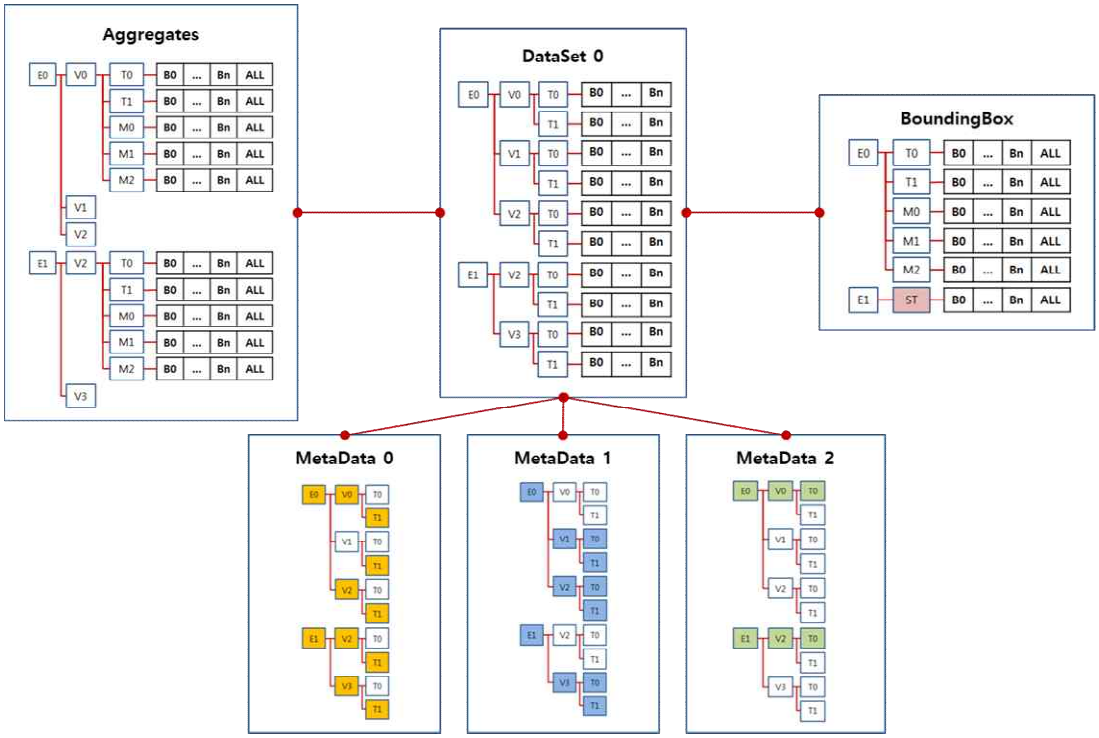
Data 디렉토리는 각 격자점이 가지고 있는 물리값을 표현한 데이터를 저장한다. 병렬 입출력 효율을 높이기 위해 변수 별로 구분해 저장한다. Mesh나 CellInfo 디렉토리와는 다르게 변수-Element-타임스텝 순서로 구성되는데, 이는 일부 변수만 선택해 로드하거나, 2차 변수를 생성하는데 편리한 구조이기 때문이다.



[그림 2] Mesh 디렉토리의 구성

CellInfo 디렉토리는 비정렬격자 데이터에만 존재하며, 셀 ID, 타입, 위치 정보를 포함하고 있고, 그 구조는 Mesh 디렉토리와 동일하다.

GLOVE 가시화 서버는 이러한 디렉토리 구조를 가지는 데이터 셋을 계산에 참여하는 전체 노드가 병렬로 읽어 들여 분산공유메모리에 적재한다. 분산공유메모리에 적재된 데이터는 [그림 3]과 같은 구조로 관리된다. DataSet에 등록된 데이터는 원본 데이터이다. MetaData는 사용자별로 선택적 데이터의 적재 시 동일한 데이터 셋이라 할지라도 타임스텝과 변수를 선택적으로 적재할 수 있으므로 이를 관리하기 위한 자료구조이다. 하나의 MetaData class의 instance는 그 DataSet을 적재한 하나의 사용자(클라이언트)에 1:1로 대응한다. Aggregates와 가시화의 대상이 되는 다양한 물리값이 가지는 통계를 관리하고 BoundingBox는 격자의 바운딩 박스에 대한 정보를 관리한다. GLOVE는 다양한 가시화 알고리즘들이 필요로 하는 간단한 통계 값들을 데이터 적재 시 계산하여 분산공유메모리에 함께 관리한다.

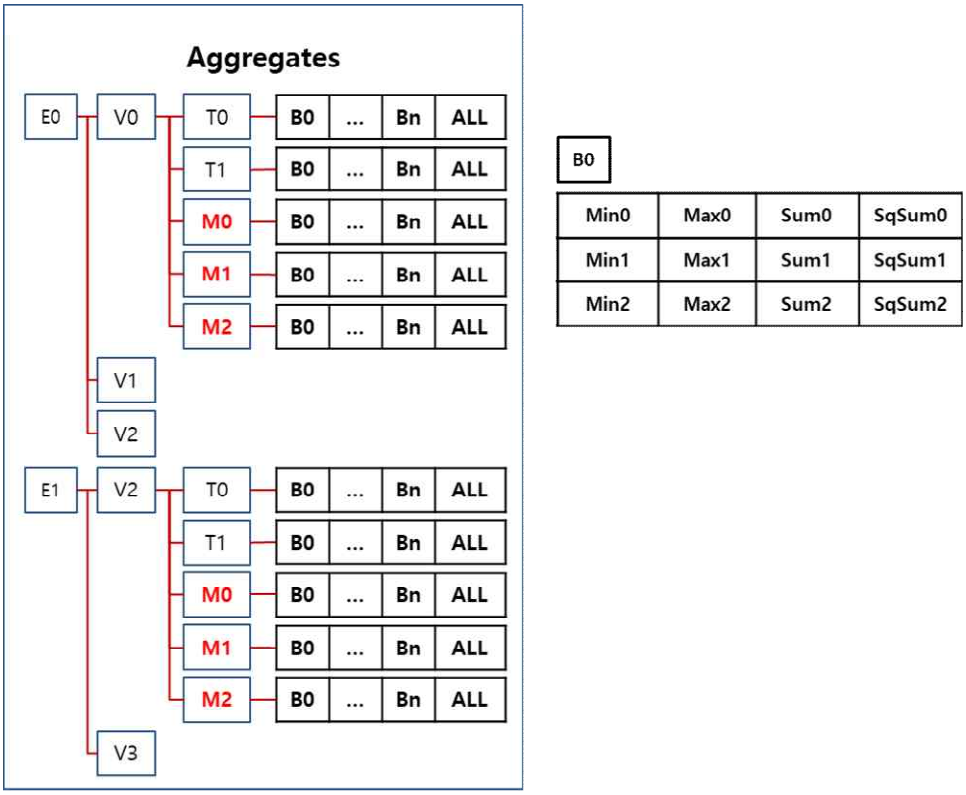


[그림 3] GDM 데이터 구조

2.3. 메타 데이터 계산

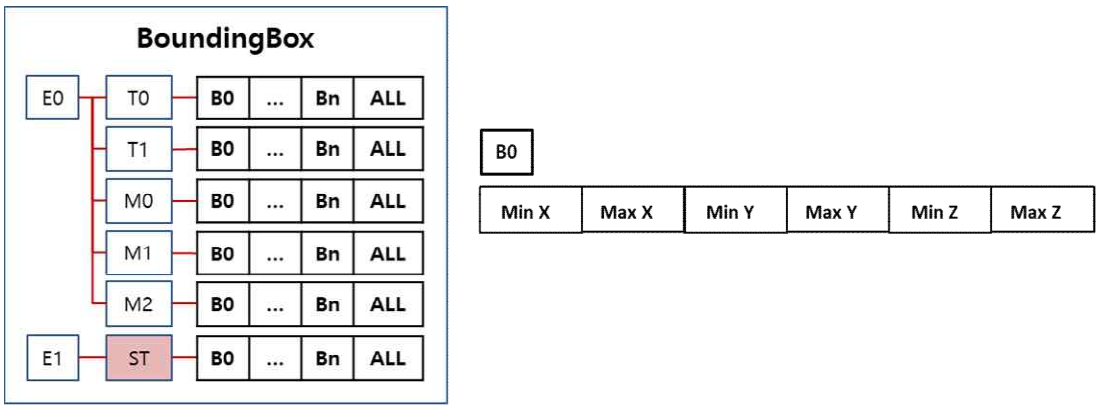
meta.xml 파일은 데이터의 형식을 기술하는 메타 데이터 파일이다. 가시화를 수행하기 위해서 필요한 데이터는 원본 데이터의 형식 외에도 실제 데이터가 가지는 값에 대한 다양한 정보를 필요로 한다. 이들 정보들은 데이터를 디스크에서 한 번 읽어 들일 때 계산하는 것이 매번 새로 계산 하는 것보다 효율적이므로 데이터 관리자는 이들 정보를 적재 시간에 계산한다. GLOVE 서버를 위해 계산하는 메타 데이터의 종류는 물리값들의 Min, Max, Sum, Squared Sum이다. 이에 더하여 가시화를 위해 데이터 적재 시 계산 되어져야 할 값은 격자의 바운딩 박스이다. [그림 4]는 Aggregates 클래스가 관리하는 통계값의 구조를 보여 주고, [그림 5]는 격자의 바운딩 박스를 위해 GLOVE가 관리하는 데이터의 구조를 보여 준다.

Aggregates는 각 Element에 해당하는 변수 V_i 별로 각 타임스텝 T_i에 대해 Min, Max, Sum, Squared Sum을 관리하고 있으며, 스칼라 변수인 경우 1차원, 다차원 벡터인 경우 차원의 수만큼 Min, Max, Sum, Squared Sum을 관리한다. 벡터의 경우 [그림 5]의 B_0와 같이 3차원으로 변수의 값을 저장한다. M_0, M_1, M_2는 사용자별 선택적 데이터 적재로 타임스텝별 value별 통계값을 달라질 것을 고려하여 전체 타임스텝에 대한 통계만 사용자별로 별도로 관리한다.



[그림 4] Aggregates 클래스의 구조

[그림 5]는 BoundingBox의 구조를 보여준다. 격자 데이터의 바운딩 박스 즉, 3차원 좌표계의 min X, max X, min Y, max Y, min Z, max Z는 가시화할 대상의 공간 좌표상의 대략적인 범위를 관리하여 알려줌으로써 가시화 알고리즘의 적용 범위를 줄여 성능 향상에 일정 부분 기여할 수 있다.



[그림 5] BoundingBox 클래스의 구조

메타 데이터의 계산 단계에서는 모든 데이터를 읽어야 얻을 수 있는 이들 데이터를 적재 단계에서 계산하는 과정을 수행한다.

2.4. 메타 데이터 분산공유메모리 반영

2.3에서 계산된 데이터는 가시화 알고리즘에 필요한 정보들이므로 분산병렬 처리 시에 전체 노드에 공유되어야 한다. 이 때문에 계산되어진 정보는 적재 완료 시 데이터 관리자가 관리하고 있는 분산공유메모리 영역에 반영해야 한다. GLOVE의 분산공유메모리 관리자는 Aggregates와 BoundingBox가 관리하는 정보와 같은 크기 같은 구조의 영역을 할당해서 계산 종료 후에 이를 반영할 수 있도록 한다. 데이터 적재에 참여하고 있는 모든 프로세스는 자신이 읽은 데이터로 만들어진 통계 데이터를 각자 해당 분산공유메모리 영역에 write 한다. 분산공유메모리 영역은 병렬 프로세스들이 올바른 write를 수행할 수 있도록 락(lock)을 걸어 계산 결과의 무결함을 보장한다.

2.5. 데이터 적재 성능 분석

GLOVE의 데이터 적재 과정은 2.3~2.4에서 살펴본 바와 같이 ①원본 데이터 I/O, ②메타 데이터 계산, ③메타 데이터 계산 결과를 분산공유메모리에 반영하는 단계로 구성된다. ①에서는 디스크에 있는 파일들을 다수의 프로세스가 병렬로 읽어 분산공유메모리에 적재하고, ②에서는 각 프로세스가 읽은 데이터에서 블록별, 타임스텝별, 변수별 최대/최소/합계/평균/분산을 계산하고, ③에서는 각 프로세스의 메타 데이터를 분산공유메모리에 반영해서 전체 데이터에 대한 메타 데이터를 만들고 이를 다시 각 프로세스로 반영한다.

기존 데이터 적재 과정의 병목 지점을 분석하기 위해 데이터 적재 과정의 세부단계별 소요 시간을 노드 수에 따라 측정한 결과는 [그림 6]과 같다. (실험 환경 및 데이터의 세부 내용은 [표 2]를 참조)

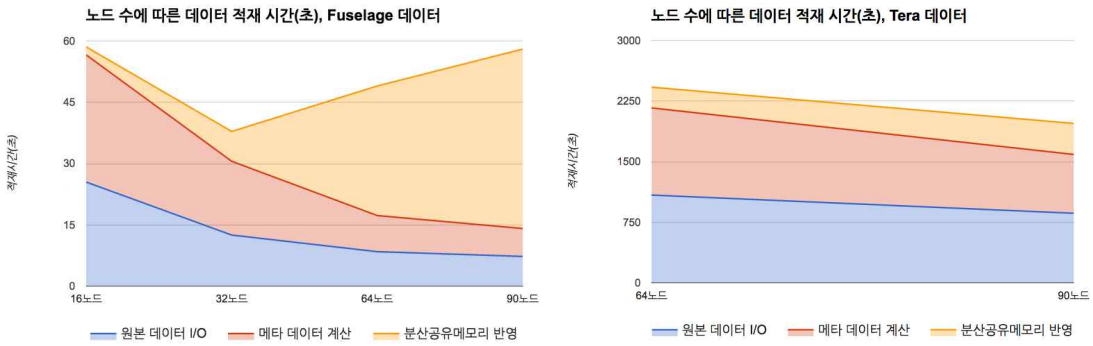
[표 2] 실험 환경 및 데이터

※ 실험 환경

노드 수		90	
CPU		Xeon X5450 3.0GHz	
노드 당 메모리 크기	32GB	전체 메모리 크기	2.88TB

※ 실험 데이터

데이터 이름	격자 종류	격자 개수	단일 타임스텝 크기	전체 타임스텝 크기
Fuselage	정렬격자	3억 개	0.2GB	6.4GB
Tera	정렬격자	1.4억 개	10.9GB	0.956TB



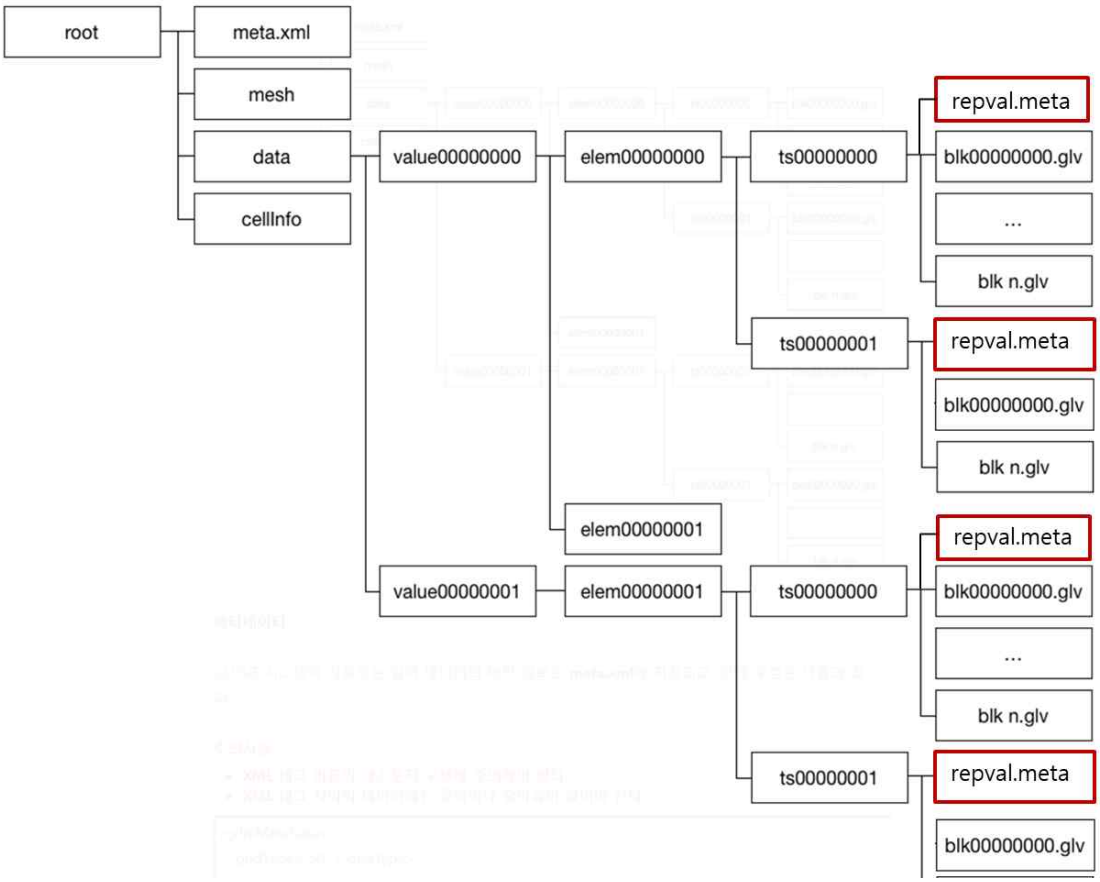
[그림 6] 노드 수에 따른 적재 시간(초)

Fuselage와 Tera 데이터 모두 공통적으로 ②메타 데이터 계산 과정에서 ①원본 데이터 I/O 단계와 비슷한 시간이 소요됨을 확인할 수 있다. 그리고 노드 수가 증가하면서 ①, ②단계 모두 소요시간이 줄어드는 경향을 보이는데 이는 병렬 처리로 얻어지는 효과이다. 하지만 왼쪽의 Fuselage 데이터의 경우, 특정 노드 수(32노드)를 초과하면 ③분산공유메모리 반영 단계의 소요시간이 급격하게 증가하여 오히려 전체 과정의 소요시간이 증가하는 양상을 보인다. 오른쪽의 Tera 데이터는 데이터가 상대적으로 크기 때문에 ①, ②단계의 소요시간에 비해 ③단계의 비중이 적어 그 효과가 작아 보이지만 역시 ③단계의 소요시간은 증가하고 있음을 확인할 수 있다. 이와 같은 ③분산공유메모리 반영 단계에서 발생하는 병목 현상은 데이터 적재를 진행하는 병렬 프로세스들이 동일한 분산공유메모리 영역을 접근하기 위해 사용하는 락(Lock)이 주요 원인이다.

3. GLOVE 메타 데이터 처리 기법 개선

3.1. 메타 데이터 계산 결과 저장

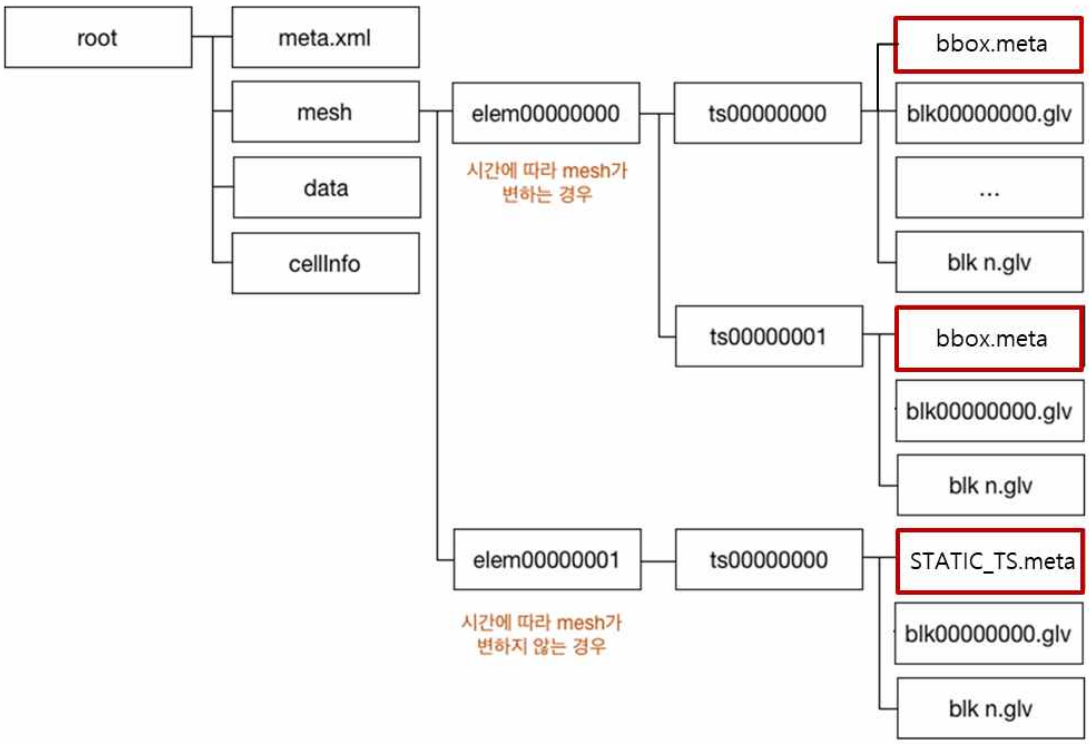
2.5에서 분석한 결과에 따르면 메타데이터의 계산 과정에 소요되는 시간이 원본 데이터의 적재 시간과 동일한 비율만큼 걸린다. 데이터가 변하지 않는다면, 필요로 하는 통계값도 변하지 않으므로 적재할 때마다 이를 계산할 필요는 없다. 따라서 한 번 적재한 데이터 셋에 대해서는 계산한 메타 데이터를 저장하여 이 과정을 생략할 수 있다. GLOVE가 계산하는 메타 데이터는 Aggregates 클래스와 BoundingBox 클래스로 관리된다. 이들 데이터는 2장에서 기술한 원본 데이터의 디렉토리 구조에서 해당 Element 및 변수, 타임스텝에 해당하는 디렉토리에 함께 저장함으로써 추후 동일 데이터 셋을 적재할 때, 필요한 정보만 다시 계산할 수 있도록 한다.



[그림 7] 물리값 별, Element 별, 타임스텝 별 통계값 저장

[그림 7]에서 실제 물리값의 통계값을 관리하는 Aggregates 클래스는 repval.meta라는 파일이름으로 해당 물리값이 실제 존재하는 디렉토리에 바이너리 형태로 해당 저장된다.

[그림 8]은 격자에 대한 통계에 해당하는 BoundingBox는 bbox.meta 파일로 해당 격자 정보가 저장되는 디렉토리에 저장됨을 보여준다. 실제 데이터 셋을 적재할 때 사용자는 데이터 셋이 저장된 가장 상위 디렉토리에 meta.xml의 위치만 알아도 해당 통계 값을 읽어 Aggregates 클래스와 BoundingBox 클래스로 적재할 수 있다.



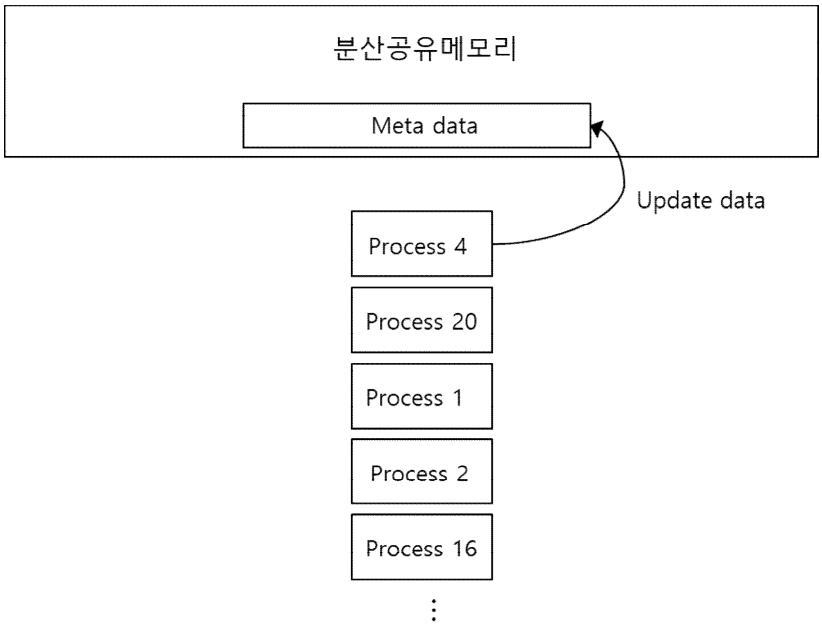
[그림 8] 격자의 바운딩 박스 데이터 저장

3.2. 분산공유메모리 반영 병목 해소

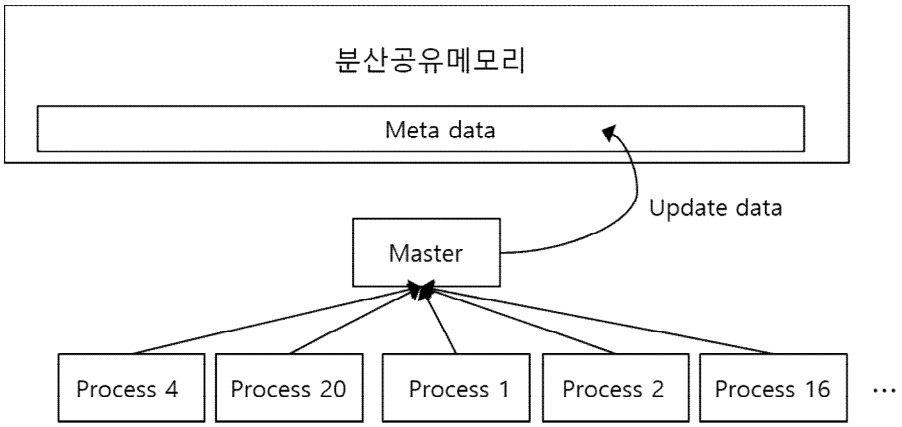
2.5에서 분석한 2가지 병목의 원인 중 하나는 계산된 통계값을 포함한 메타 데이터 정보를 분산공유메모리에 반영하는 단계에서 다수의 프로세스들이 동일한 분산공유메모리 영역을 접근하기 위해 락(Lock)을 사용하면서 발생하는 성능 저하 문제였다. 본 문서에서는 이를 개선하기 위해 분산공유메모리에 접근하는 프로세스의 수를 최소화 하는 기법의 구현을 기술한다.

기존의 통계 계산을 위한 접근 방식은 [그림 9]와 같이 분산공유메모리 데이터 영역에 하나의 통계 계산을 위한 영역을 만들고 데이터를 디스크로부터 읽어서 계산을 수행하는 모든 프로세스가 자신이 읽어야할 데이터를 다 읽은 시점에 자신이 계산한 값을 미리 만들어진 분산공유메모리 영역에 반영하는 방식으로 이루어졌다. 이 때문에 데이터를

읽고 더하는 과정 전체에 락을 걸어야 해서 데이터를 모두 읽을 시점에 대부분의 프로세스가 자신의 통계 데이터를 분산공유메모리 영역에 쓰기위해 대기함으로써 다른 일을 하지 못하여 병렬 효율이 떨어지는 구조였다. 이러한 문제점을 해결하기 위해 데이터를 적재하는 데이터 관리자는 데이터를 적재하는 모든 프로세스가 통계값을 업데이트 하도록 하지 않고 [그림 10]과 같이 하나의 프로세스가 전체 통계를 취합하도록 하였다. 수정된 방식은 각각의 프로세스들이 자신이 계산한 값을 마스터 프로세스에게 전달하고 나면 다른 작업을 수행할 수 있으므로 병렬 효율성이 높아져 적재 성능 향상을 기대할 수 있다.



[그림 9] 기존의 메타 데이터 반영 방식

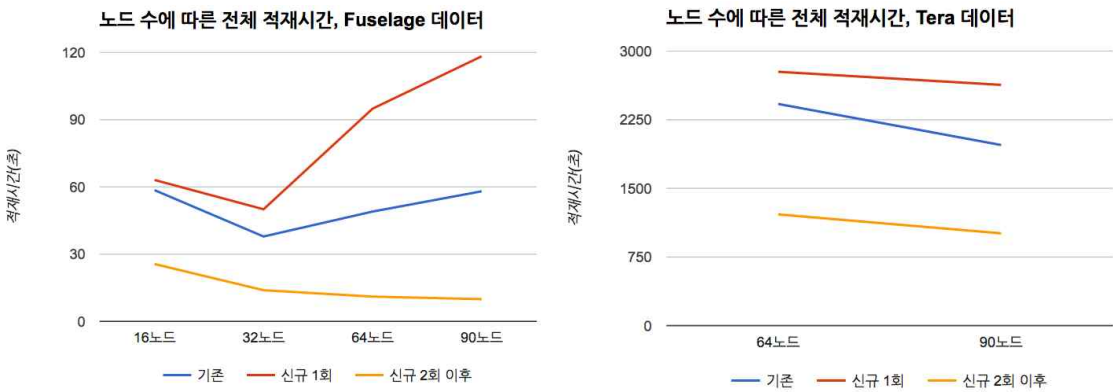


[그림 10] 새로운 메타 데이터 반영 방식

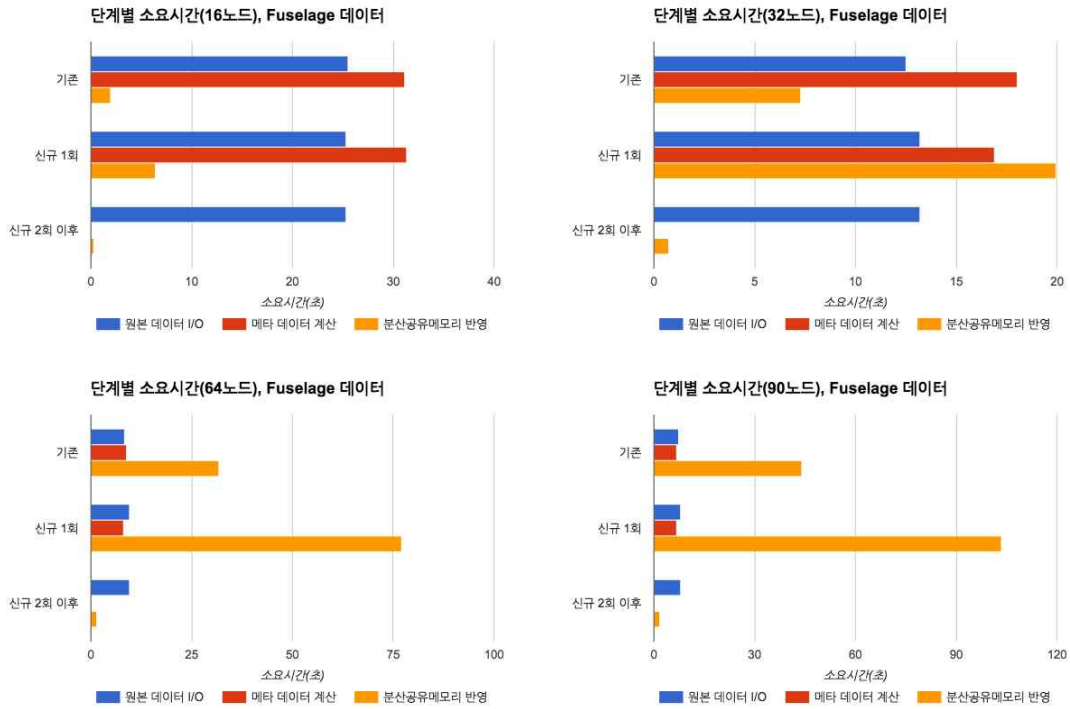
4. 실험결과 및 성능 확인

적용한 기법의 성능 개선을 확인하기 위해 Fuselage, Tera 데이터에 대하여 노드 수를 16, 32, 64, 90으로 늘려가며 실험을 수행하여, 적재 시간을 측정하여 기존 방식과 비교하였다. [그림 11]의 결과는 신규 방식은 처음 수행하는 적재 시간은 적재 단계에서 계산한 결과를 파일로 저장하는 시간을 포함하므로 기존 데이터의 적재시간보다 긴 시간이 걸리지만 2회 이후부터는 절반 이상으로 줄어듦을 보여 준다. [그림 12]와 [그림 13]에서 세부단계별 소요시간을 살펴보면 신규 방식의 2회 이후부터는 ②메타 데이터 계산 단계가 생략되고, ③분산공유메모리 반영 단계의 소요시간이 절반 수준으로 감소하여 100% 성능 향상을 거두었음을 확인할 수 있다.

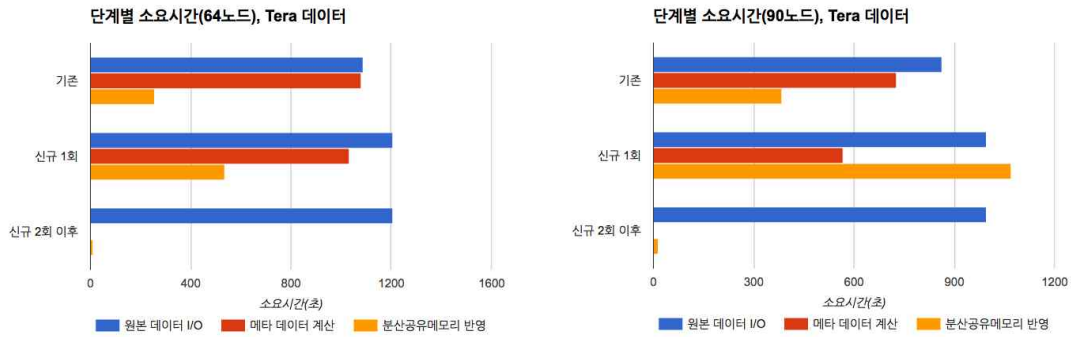
	Fuselage (6.4GB)			Tera (0.956TB)		
	기존	신규		기존	신규	
		1회	2회 이후		1회	2회 이후
16노드	58.56	63.09	25.60 (229%)	-	-	-
32노드	37.87	50.02	13.92 (272%)	-	-	-
64노드	49.00	94.90	11.06 (443%)	2421.75	2773.11	1215.58 (199%)
90노드	58.04	118.27	9.93 (585%)	1973.43	2630.07	1008.79 (196%)



[그림 11] 노드 수에 따른 전체 적재시간의 비교



[그림 12] 노드 수에 따른 단계별 소요시간(Fuselage 데이터)



[그림 13] 노드 수에 따른 단계별 소요시간(Tera 데이터)

5. 결론

본 문서에서는 대용량 유동 해석 데이터 가시화 도구인 GLOVE의 데이터 적재 성능 향상을 위한 기법들에 대해 살펴보았다. 데이터 적재 시간의 많은 부분을 차지하는 메타 데이터 계산 시간을 줄이기 위해 기존 계산 결과를 활용하여 저장하고 해당 정보의 분산공유메모리 반영을 위해 다수의 병렬 프로세스가 동시에 쓰는 것을 방지하기 위해 사용하는 락(lock)으로 인해 발생하는 지연을 줄이기 위해 하나의 프로세스가 분산공유메모리에 쓰도록 구현함으로써 전체 적재 시간을 100% 향상하였다.

그러나 여전히 존재하는 최초 데이터 적재 시 [그림 11]에서 최초 1회 적재 그래프가 보여 주는 것과 같이 데이터 사이즈가 비교적 작고 노드수가 많을수록 급격히 증가하는 적재 시간을 줄이기 위해 데이터 사이즈가 적고 노드 수가 많을 때, 컴퓨팅 노드와 프로세스를 적응형으로 할당하는 방안과 병렬 파일 시스템의 최대 수용 가능 병렬 노드의 수에 따라 적재를 수행할 노드와 프로세스의 수를 제한하는 기법의 개발 등이 추가로 수행되어져야 할 것이다.

6. 참고문헌

[1] Min-Ah Kim et al, “GLOVE(GLObal Virtual reality visualization Environment for scientific simulation): VR환경에서의 대용량 데이터 가시화 시스템“ 2010 한국컴퓨터종합학술대회 논문집 제37권 제2호(B), 267-271

[2] Vishwanath, V., “LambdaRAM: A High-Performance, Multi-Dimensional, Distributed Cache Over Ultra-High Speed Networks“, Dec.15, 2008.