ISBN 978-89-6211-031-9 93560

클러스터파일시스템 I/O 성능 분석 보고서

우 준

<제목 차례>

<표 차례>

<그림 차례>

제1장 개요

고도의 연산을 요하는 과학응용 연산을 수행하기 위하여 슈퍼컴퓨터와 같은 고가의 단일 기종을 사용하는 대신, 저가의 다수 연산 노드(processing node)들을 고속의 네트워크를 통해 하나로 묶는 클러스터 시스템은 이미 수퍼컴퓨팅 분야에서 상당한 부분을 차지하게 되었다.

클러스터 시스템은 기존의 단일 슈퍼컴퓨터들에 비해 가격 면에서 앞설 뿐 아니라, 성능(throughput), 확장성(scalability), 가용성(availability)등 여러 면에서 우수하다. 따라서 이러한 클러스터 시스템은 기존의 대형 기종이 담당하고 있던 과학응용연산 분야에서 기존의 대형시스템 들을 대체하고 있으며, 확장성 및 높은 성능이 요구되는 웹 서버나 멀티미디어 서버 등과 같은 응용분야에서도 사용되어지고 있다. 처음 시스템을 구축할 때 충분히 작업부하와 처리량 등을 고려하여 시스템을 구성하게 되는데, 클라이언트의 수가 늘어나고 처리해야 할 데이터의 양이 증가하는 일이 발생할 수 있다. 이 때 기존의 시스템을 용량이 큰 다른 시스템으로 대체하는 것은 가격 뿐 아니라 시간 등에 많은 시간과 노력이 소요된다. 그러나 클러스터 시스템에서는 시스템에 노드를 하나씩 추가함으로써 낮은 가격과 손쉽게 전체 시스템의 성능을 높일 수 있는 확장성을 제공한다.

클러스터 파일시스템은 이를 구성하는 여러 개의 노드들이 고속의 네트워크로 연결되어 있다는 점에서 분산파일시스템과 유사하고 여러 개의 노드에서 요구하는 입출력을 동시에 처리한다는 점에서 병렬파일시스템과 유사한 점이 있다. 분산파일시스템이 주로 한대의 파일시스템서버가 공유에 사용되는데 반하여 클러스터 시스템은 노드들 간에 유기적으로 협력하여 작업을 처리

한다.

본 연구에서는 대표적인 클러스터 파일시스템인 Lustre와 GPFS의 주요 특징을 분석하고 I/O 성능을 분석하였다.

제2장 주요 클러스터 파일시스템 특징

1. Lustre

Lustre라는 이름은 Linux 와 Cluster의 합성어로서, Lustre File System은 오픈소스 기반의 고성능 클러스터 파일시스템이다. 기존의 NFS 같은 분산 파일시스템에 클러스터기술과 Object-based Storage Disk 기술을 적용함으로써 성능, 가용성, 확장성의 문제를 개선하였다.

Lustre는 기존의 하드웨어 위에서 동일하게 설치하여 구동할 수 있으며, OSD(Object-based Storage Disk) 기술을 사용하여, 파일 I/O 메타데이터 연산을 따로 분리하였다. 이러한 파일시스템 디자인을 통해서 파일 I/O 연산을 분리시킴으로써 더 효율적인 자원 활용을 가능하게 한다.

가. Object-based Storage Disk

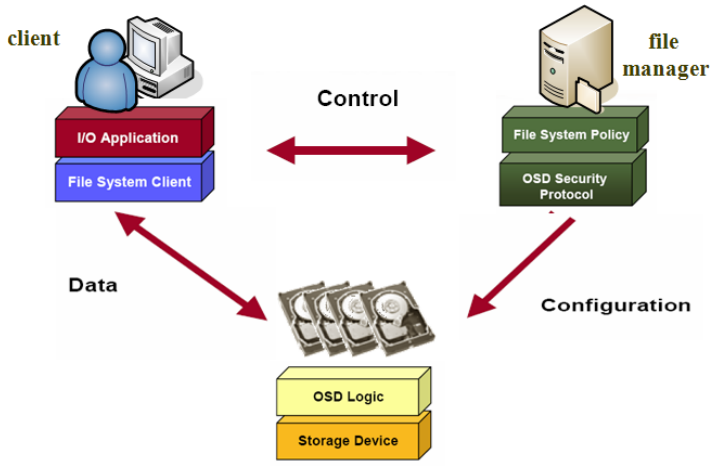
Lustre 파일시스템에서는 기존의 파일과 블록기반의 인터페이스를 사용하는 공유 파일시스템이 갖는 단점을 극복하기 위해서 객체 기반의 인터페이스를 사용하여 확장성, I/O 성능, 보안성을 동시에 만족시켰다.

기존의 NFS 같은 파일인터페이스 기반 공유파일 시스템은 사용자에게 블록을 파일단위로 추상화하여 제공하므로 액세스 컨트롤을 가능하게 하여, 보안적인 측면을 제공하지만, 파일서버는 단 한대이기 때문에, 동시에 다수의

클라이언트에서 I/O를 요청할 경우에 성능의 저하를 피할 수 없다.

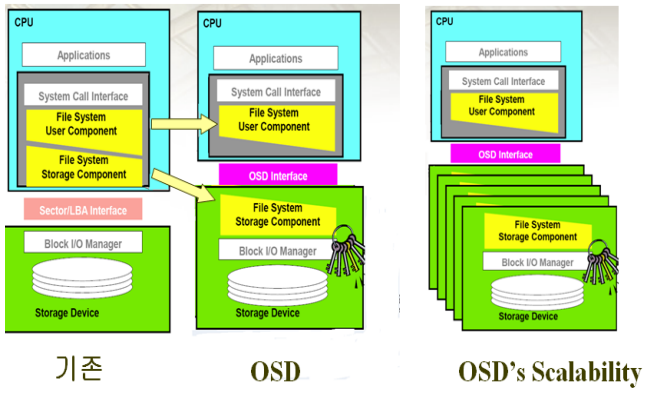
SAN과 같은 블록인터페이스 기반 공유파일 시스템은 성능과 확장성이 매우 뛰어나지만 I/O를 관리하고 메타데이터를 유지하는 파일서버가 없이는 데이터 공유와 보안 측면에서 한계가 있다.

이것을 해결하기 위해서 OSD에서는 파일-블록 매칭 연산을 OS의 역할에서 디스크의 역할로 분리시키고, 파일 I/O를 위한 공통적인 인터페이스를 디스크에 내장시켰다. 객체지향 프로그래밍에서 객체의 인터페이스를 통해서 기능을 사용하듯이, OSD에서도 공동적인 인터페이스를 통해 모든 OSD에 접근 가능하다. <그림2-1>에서 보듯이 파일을 쓰거나 읽기를 원하는 클라이언트는 자신이 원하는 파일이 어떤 OSD에 있는지 파일 매니저를 통해 메타데이터를 얻어옴으로서 알 수 있게 되고, 이후 원하는 데이터 액세스를 위해 자신이 원하는 데이터가 있는 OSD 들에게 공통적인 인터페이스를 사용하여 접근한다.



<그림 2-1> OSD 구조

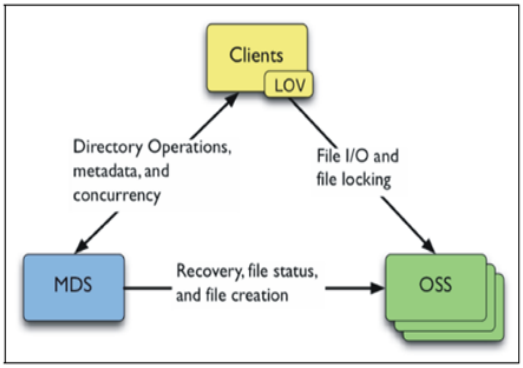
OSD가 공통적인 인터페이스를 제공함에 따라 클라이언트들은 다수의 OSD에 동시에 접근할 수 있게 되고, 이에 따라 I/O 성능과 확장성이 개선된다. 또한 중앙의 파일 매니저서버를 통해서 액세스컨트롤이 가능하기 때문에 보안성 또한 개선된다. <그림 2-2>는 기존의 파일-블록 맵핑 기능을 OS에서 디스크로 옮겨 디스크에 객체단위로 저장하도록 하는 구조와, 객체 단위로 저장함으로 인해 모든 OSD에 공통적으로 접근하여 확장성이 개선됨을 보여주고 있다.



<그림 2-2> OSD 확장성

나. Lustre 클러스터 구성

Lustre 파일 시스템은 <그림2-3>에서처럼 크게 Meta Data Server(MDS), Object Storage Server(OSS), 클라이언트 세부분으로 이루어진다.



<그림 2-3> Lustre 구성

Lustre 파일시스템에는 OSD를 Lustre에 적용하기 위해서 일반적인 블록기반 디스크(ex. HDD)가 컴퓨터의 도움을 받아 OSD의 역할을 한다. 파일매니저의 역할을 수행하도록 도움을 주는 컴퓨터를 MDS, OSD의 역할을 할 수 있도록 도움을 주는 컴퓨터를 OSS라고 부르게 된다. 하나의 OSS에는 여러 개의 블록기반 디스크가 물리적으로 연결되어 각각의 디스크마다 OSD로서의 역할을 수행 할 수 있으며, 이 각각의 디스크를 OST(Object Storage Target) 라 부르게 되고, MDS에 연결된 각각의 디스크를 MDT(MetaData Target) 라고 부른다

MDS는 메타데이터를 클라이언트들에게 제공해준다. Lustre 파일 시스템은 성능과 확장성을 개선하기 위해 파일을 MDS를 통해서 오브젝트 단위로 다루게 되는데, MDS는 파일 생성과 검색, 파일 속성 변경, OST로의 파일 I/O 리디렉션 등의 메타데이터 연산을 수행을 한다. 클라이언트가 I/O 연산을 수행하기 위해서는 먼저 MDS에서 메타데이터 연산을 수행하여 액세스하기 원하는 파일이 저장된 OST에 대한 정보를 얻은 후, OST에 I/O 를 요청할 수 있다. 또한 MDS는 파일시스템 메타데이터변화와 클러스터상태에 대한 기록을 저장해서 하드웨어나 네트워크의 이상으로 인한 복구를 할 수 있도록 해준다.

OSS는 클라이언트와 물리적인 디스크 사이에서 I/O 요청을 처리하고, OSS를 통해서 데이터가 저장되고 읽혀진다. OSS는 내부적으로 리눅스의 파일시스템을 사용함으로 인해서 기존 파일시스템의 장점들을 그대로 살리는 한편, 기존 파일시스템의 개선이 또한 Lustre 파일시스템의 개선으로 이어질 수 있도록 하였다.

Lustre 파일 시스템은 구동 중에 새로운 OST를 동적으로 추가가 가능하며, 새로 추가된 OST는 자동적으로 MDS에 등록되어 사용가능하다.

다. LOV(Logical Object Volume)

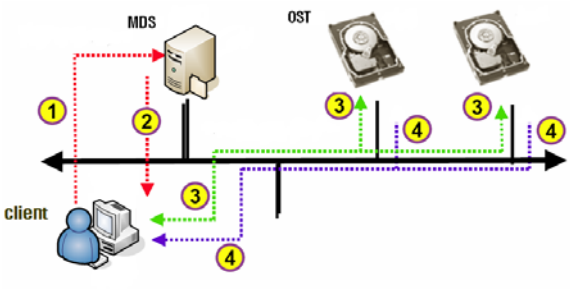
LOV는 Lustre 클라이언트들에게 하나의 마운트 가능한 장치로 보이는 단위이다. Lustre 파일 시스템은 하나의 MDT와 다수의 OST를 하나로 묶어서 하나의 파일시스템을 제공하며, 클라이언트들은 모두 LOV의 이름을 사용하여 접근가능하다.

MDT에 대한 장애복구 기능을 사용할 경우 MDT를 2개로 구성하여, MDT 서버에 장애가 일어나 전체 파일시스템에 액세스가 불가능하게 되는 상황을 방지 할 수 있다.

라. Lustre 파일 전송

Lustre의 파일전송은 크게 두가지 단계로 나뉜다. <그림 2-4>에서 먼저 클라이언트는 ①자신이 원하는 파일의 부분을 저장하고 있는 OST의 주소를 MDS에 요청하고, ②MDS로부터 자신이 원하는 파일이 있는 OST의 주소를

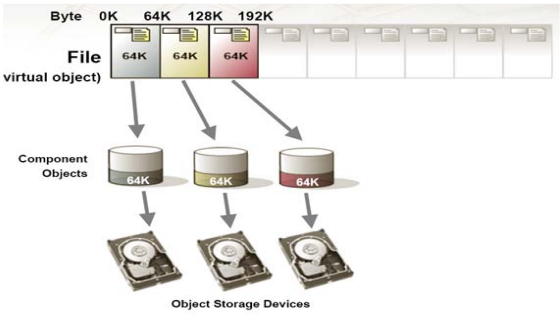
받게 되면, 이후에는 MDS와의 통신 없이 OST와만 직접적으로 I/O를 계속해서 수행하게 되며, ③MDS로부터 받은 메타데이터를 바탕으로 OST에 I/O 요청을 하면, ④OST와의 데이터 전송이 시작된다.



<그림 2-4> 파일전송 흐름도

마. 파일 스트라이핑(striping)

Lustre 파일 시스템은 MDS에 파일 시스템의 OST와 파일 스트라이핑 개수와 블록크기에 대한 정보가 함께 포함되어 있어, 파일을 저장할 때 자동적으로 다수의 OST에 분할하여 병렬 처리함으로써 I/O 성능을 극대화한다. <그림 2-5>는 파일을 192KB의 파일을 64KB 단위로 3개씩 나눠서 저장하는 것을 보여준다.



<그림 2-5> 파일 스트라이핑

바. Lustre 지원 환경

Lustre는 다양한 플랫폼과 리눅스위에 설치하여 구동 할 수 있고, 대부분의 네트워크 인터페이스를 지원한다.

<표 2-1 > Lustre 지원 환경

운영체제	Red Hat Enterprise Linux 3+ SuSE Linux Enterprise Server 9 and 10 Linux 2.4 and 2.6
플랫폼	IA-32, IA-64, x86-64, PowerPC
네트워크	Ethernet, Quadrics Elan3 4, Myrinet, Mellanox, InfiniBand

사. Lustre v1.6에서 추가된 특징

- 서버 환경설정

  ManaGement Server(MGS)를 추가함으로써 v.1.4에서의 각 서버 및 클라이언트 마다 동일한 configuration 파일을 복사해야 하는 불편함을 해결하였다.

- MGS 서버

  MDS, OSS, LOV에 대한 모든 정보를 소유하며, 각 MDS 및 OSS서버, 클라이언트 들은 MGS에 접속하여 configuration 정보 수신

2. GPFS

IBM의 GPFS(General Parallel File System)는 다수의 노드로부터 기존의 POSIX API를 통해 동시적인 파일액세스를 가능하게 하는 블록 I/O 기반의 클러스터파일 시스템이다. GPFS는 기존의 분산 파일시스템의 I/O 대역폭 이상의 성능을 내도록 설계되어, 큰 데이터 대역폭이 요구되는 환경에 적합하다.

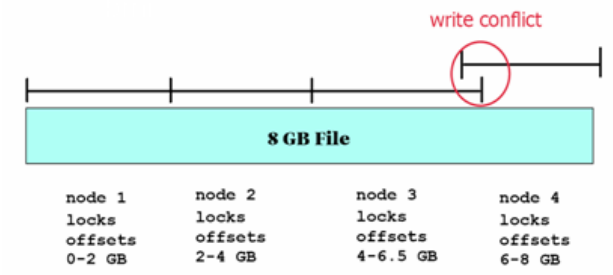
GPFS는 리눅스와 AIX의 OS를 지원하며, 다양한 아키텍처를 기반으로 하는 시스템을 지원하여 클러스터 환경에서 다양한 특성을 가진 노드들이 각각의 파일시스템을 공유하여 접근 할 수 있는 뛰어난 호환성과 확장성을 가진다. 또한 메타데이터 서버를 별도로 구성할 수 있도록 허용하여 파일시스템에 대한 메타데이터 액세스가 증가 할 때 메타데이터 처리에 대한 병목현상을 해소하였다.

클러스터 내의 클라이언트 노드마다 파일 I/O를 위한 GPFS 전용 버퍼인 Pagepool을 할당하여 시스템의 상황에 영향을 받지 않고 일관성 있는 I/O성능을 유지 할 수 있으며, 블록의 공간 효율성을 위해서 하나의 블록을 32개의 부분블록으로 나누어 사용한다.

또한 안정성을 위해서 데이터와 메타데이터의 복사본을 만드는 것이 가능하며, 서버에 대한 장애복구 구성이 가능하다.

클러스터 환경에서는 동시에 다수의 클라이언트들이 하나의 파일을 사용하는 경우가 많은데 이를 위해서 바이트 단위의 Lock 알고리즘을 사용한다.

이것은 파일 I/O를 할 때 기존의 파일전체를 Lock 하는 방식에서 파일에서 필요한 부분만 바이트 단위로 Lock을 획득함으로서 동시에 같은 파일을 write 할 수 있도록 하여 같은 파일에 대한 병렬 처리를 수행한다. <그림 2-8>에는 8GB 파일 하나에 대해서 동시에 세 개의 노드가 부분적으로 Lock을 획득하고 있는 것을 보여주며, Lock을 획득하려는 영역이 충돌되는 노드4만 Lock을 획득하지 못하고 있다.



<그림 2-6> 바이트 단위 Locking

가. GPFS 노드 구성

GPFS 클러스터를 구성하는 하나의 노드는 다음의 4가지로 구성된다.

(1) GPFS 관리 명령어(GPFS administration commands)

이 관리 명령어를 통해서 GPFS 클러스터의 어느 노드에서나 GPFS 환경설정과 관리 작업을 수행할 수 있다.

(2) GPFS 커널 확장(The GPFS kernel extension)

GPFS 커널 확장은 운영체제의 VFS에 GPFS 파일시스템에 대한 인터페이스를 제공한다. 응용프로그램이 운영체제에 시스템 콜을 호출할 때, 호출된 시스템 콜은 VFS를 통해서 GPFS 커널 확장에 도착하여 처리한다. GPFS 커

널 확장을 통해서 응용에 투명하게 파일시스템으로서 동작한다.

(3) GPFS 데몬(The GPFS daemon)

GPFS 데몬은 모든 I/O와 버퍼 관리를 수행한다. 순차적인 read 작업에 대해서 read-ahead를 수행하고, 동기적으로 명시되지 않은 write에 대해서 write-back을 수행하며, lock token 관리를 통해서 파일시스템의 일관성을 유지한다.

다른 노드에 있는 데몬과 서로 통신함으로써 GPFS 클러스터 파일시스템의 환경설정을 공유하고, 장애복구 기능을 수행한다.

(4) 호환성 계층(The GPFS open source portability layer)

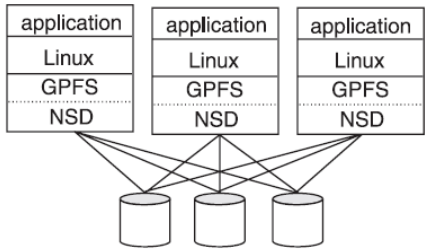
리눅스 커널과 GPFS 모듈의 통신을 위해서 필요하며, 몇가지 특별한 플랫폼과 리눅스를 위해서 필요하다.

나. GPFS 클러스터 구성

GPFS 클러스터에 포함되는 노드들은 NSD(Network Shared Disk)라고 하는 가상의 디스크를 갖게 되는데, 클러스터 안에서 NSD는 고유한 이름을 가지고 노드들은 이를 통해 접근하게 되고, 모든 노드들은 NSD로 구성된 파일시스템을 직접 마운트 함으로서 같은 공간에 접근이 가능하게 된다. NSD는 물리적으로 연결되어져 있는 경우와 네트워크를 통해 다른 NSD 서버의 도움을 받아 연결되어져 있는 경우로 나뉘어 지는데, I/O를 시도할 때 NSD는 자신의 연결 상태를 확인하여 적절한 방법으로 파일시스템을 액세스 한다.

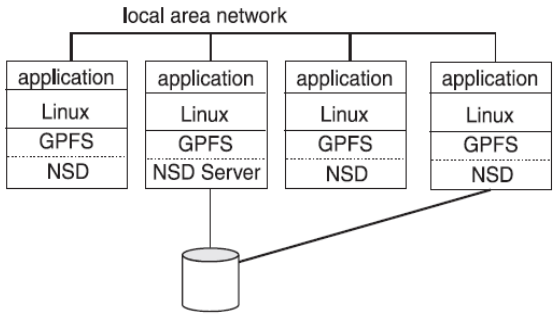
Lustre와 달리 GPFS는 메타데이터모듈과 데이터처리모듈을 따로 분리하지 않고, 기본적으로 모든 서버노드에서 데이터와 메타데이터 처리를 동시에 하며, 메타데이터 서버를 따로 구성하기를 원하는 경우에는 메타데이터만을 위한 서버를 따로 지정할 수 있다.

다음은 GPFS 서버의 가능한 구성이다. GPFS는 Linux와 AIX 클러스터를 모두 지원하며, Linux와 AIX간에도 NSD 서버를 통해서 I/O를 할 수 있다. <그림 2-7>은 리눅스로만 구성되어 있는 GPFS 클라이언트로서 모든 NSD 노드가 SAN디스크에 물리적으로 연결되어 있어 I/O의 요청이 있을 때 물리적으로 직접 접근하게 된다. 따라서 네트워크를 통한 접근이 없으므로 NSD 서버는 존재하지 않는다.



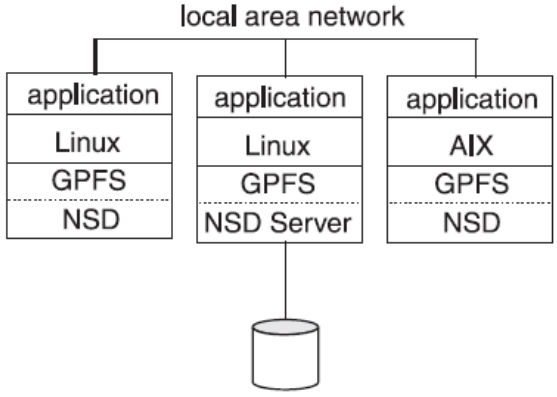
<그림 2-7> SAN디스크가 연결된 리눅스 클러스터

<그림 2-8>은 하나의 NSD노드만 물리적으로 디스크와 연결되어 있어서 네트워크를 통해 I/O 서비스를 제공하기 위해 NSD 서버로서의 역할을 수행하고, 다른 노드들은 네트워크를 통해서 디스크에 접근을 하게 된다. NSD 서버의 역할을 수행하는 노드에 장애가 일어났을 경우를 대비하여 물리적인 연결을 하나 더 추가하여 NSD 서버에 대한 백업노드로 지정할 수 있으며 가장 우측의 연결되 있는 NSD가 이것을 보여주고 있다.



<그림 2-8> 네트워크로 연결된 리눅스 클러스터

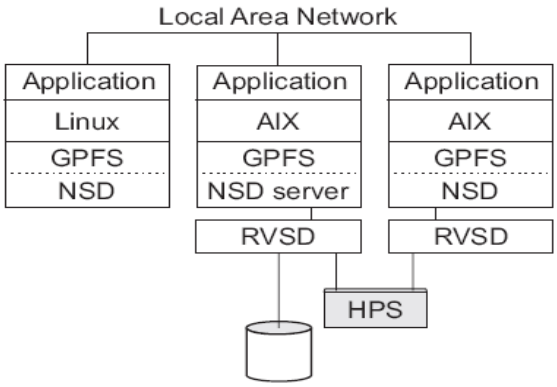
<그림 2-9>는 리눅스와 AIX가 함께 구성되어 있는 노드가 NSD 서버를 통해서 리눅스 노드에 연결되어 있는 디스크에 접근이 가능함을 보여준다. 이것은 다양한 환경으로 구성되어 있는 클러스터 환경에 호환성과 확장가능성을 높여준다.



<그림 2-9> 리눅스와 AIX 클러스터- GPFS서버:리눅스

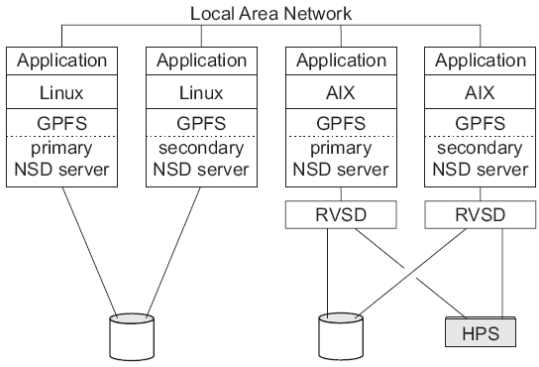
<그림 2-10>은 RVSD를 갖는 AIX 노드 NSD 서버에 다른 AIX 노드와 리눅스 노드가 네트워크를 통해서 접근 하는 구성을 보여준다. <그림 2-9>에서 리눅스 NSD 서버에 AIX 노드가 접근 할 수 있는 것처럼 AIX NSD 서버에도

리눅스 노드가 접근 가능하다. AIX NSD 서버도 HPS 스위치를 통해 디스크에 대한 직접 연결을 추가하여 백업노드를 구성할 수 있다.



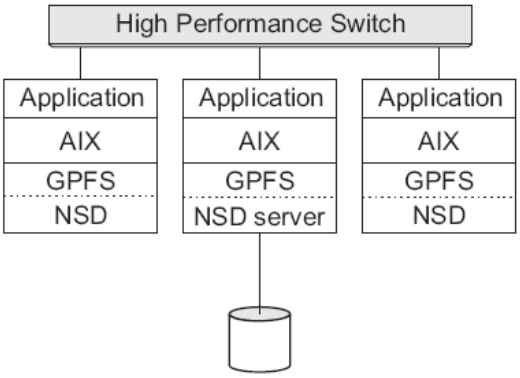
<그림 2-10> 리눅스와 AIX 클러스터 - GPFS서버:AIX

<그림 2-11>은 리눅스와 AIX 노드가 모두 각각의 물리적인 디스크를 가지고 NSD서버로 구성되어 있는 것을 보여주고 있으며, 상호간에 네트워크를 통해서 접근이 가능하다.



<그림 2-11> 리눅스와 AIX 클러스터 - GPFS서버:리눅스,AIX

<그림 2-12>는 IBM의 HPS(High performance Switch)를 통해 연결한 AIX 노드의 구성이며 하나의 노드가 NSD 서버로서의 역할을 수행하며 나머지 노드들은 HPS를 통해서 디스크에 접근한다.



<그림 2-12> HPS로 연결된 AIX 클러스터

다. GPFS 파일 전송

GPFS의 파일전송도 Lustre와 비슷하게 파일에 대한 메타데이터를 가지고 있는 노드를 파일시스템 매니저로부터 받아온 후 메타데이터를 얻어오고 그에 따라 데이터 액세스가 일어나게 된다.

Write 연산이 일어나는 과정을 살펴보면 액세스하고자 하는 파일에 대한 메타데이터를 얻어 온 후 Lock을 위한 토큰을 획득한다. 토큰 획득에 성공하면 GPFS 전용 버퍼인 Pagepool에 Write할 데이터를 전송하고, GPFS 데몬에 의해서 네트워크로 전송되어 NSD 서버에 저장 된다.

라. GPFS 지원 환경

GPFS는 다양한 플랫폼과 리눅스 및 AIX 위에 설치하여 구동 할 수 있고, 대부분의 네트워크 인터페이스를 지원한다.

<표 2-2> GPFS 지원 운영체제

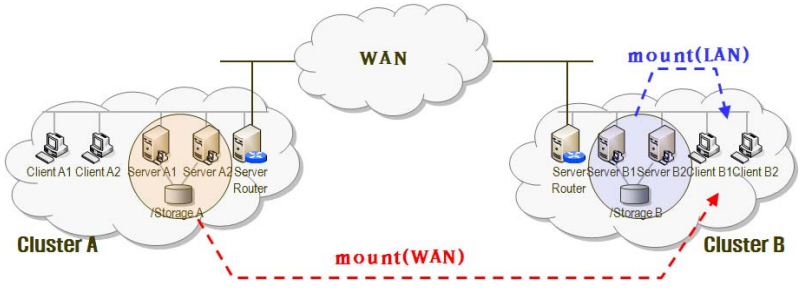
OS	AIX 5L v5.3, AIX 5L v5.2, Red Hat Enterprise Linux 4 and 5 SuSE Linux Enterprise Server 9 and 10
플랫폼	i386, x86_64, PowerPC
네트워크	Ethernet, 10GB Ethernet, Myrinet, InfiniBand, HPS, SP Switch2

제3장 I/O 성능 분석

클러스터 파일시스템은 네트워크를 통해 분산되어 있으며, 통상 LAN 영역 안에서 고성능 네트워크를 통해서 하나의 파일시스템 클러스터로 구성된다. 또한 이렇게 LAN 영역 안에서 구성된 하나의 파일시스템 클러스터는 WAN을 통해서 원거리에 있는 다른 LAN 영역 안의 파일시스템 클러스터와 연결되어 서로의 파일시스템을 공유할 수 있다.

따라서 WAN으로 연결되어 있는 클러스터간의 파일시스템 성능은 LAN으로 구성된 클러스터 파일시스템의 I/O성능을 기반으로 하여 중간의 네트워크 구성요소들의 영향을 받아 결정되므로 WAN으로 연결된 클러스터 파일시스템의 성능을 분석하기 위해서는 LAN 영역안의 클러스터 파일시스템의 I/O 성능 분석이 선행되어야 한다.

이번 장에서는 이후 다룰 WAN 클러스터 파일시스템의 I/O 성능 분석을 하기에 앞서 LAN 클러스터 파일시스템의 I/O성능 분석 결과를 보인다.



<그림 3-1> LAN / WAN 클러스터 파일시스템

1. 성능측정 방법

파일시스템의 성능을 측정하기 위해서 파일시스템 상에서 주로 사용하게 될 응용프로그램의 워크로드를 파악하여 이에 따라 I/O 성능을 측정해야 하며, 이에 따라 성능을 최적화하기 위한 구성이 달라지게 된다.

워크로드에 따른 I/O 성능 측정을 위한 가장 좋은 방법은 I/O 클라이언트에 실제로 사용될 응용프로그램을 설치하여, 응용프로그램의 주요 I/O 연산을 일으키는 것이지만 현실적으로 이런 방식으로 성능을 측정하기에는 어려움이 있다.

따라서 응용프로그램의 워크로드를 모델링하여 시뮬레이션을 통해 I/O 성능을 측정을 하게 된다. 기본적으로 워크로드를 모델링하기 위한 주요한 요소는 I/O 액세스 패턴, 블록크기, 파일크기 등이 있다.

I/O 액세스 패턴은 크게 Sequential, Strided, Random 패턴이 있다. 대용량의 벌크 I/O 같은 경우 주로 Sequential 한 경우가 많이 나타나며, 연속된 구조체 배열에서 특정한 부분만 계속적으로 액세스 할 경우에 Strided 패턴에 나타난다. Sequential 패턴에 비해서 Strided 와 Random 패턴의 경우 하드디스크의 플래터가 물리적으로 움직이는 시간이 늘어나기 때문에 I/O 성능이 저하된다.

블록크기는 한번의 I/O 요청마다 발생되는 데이터 전송크기를 말하며, 블록의 크기가 커질수록 같은 크기의 데이터에 대해서 처리해야 하는 I/O 요청이 감소하므로 I/O 성능이 향상되며, 블록크기가 파일시스템의 블록크기보다 작을 경우 Rewrite를 하는 경우 파일시스템의 블록을 읽어 온 후 Rewrite 해야

하는 부분의 메모리에 쓰고 다시 디스크에 써야하므로 성능이 저하된다.

파일크기는 크기의 변화에 따른 디스크 I/O의 성능 측정 보다는 파일시스템을 구성하고 있는 시스템의 캐시 효과를 주로 측정하게 된다. 일반적으로 I/O를 일으키는 클라이언트의 메모리 크기보다 작은 파일에 대해서는 Reread 연산의 속도가 개선된다.

클러스터 파일시스템의 경우에는 다수의 파일서버로 구성되어 있어서, 하나의 클라이언트로는 파일서버의 모든 성능을 활용 할 수 없기 때문에 클라이언트의 증가에 따른 I/O 성능과, 반대로 클라이언트가 급격하게 증가하여 서버에 가중되는 부하에 따른 I/O 성능이 워크로드를 모델링하는 요소가 될 것이다.

이 장에서는 I/O성능을 측정하기 위해서 파일시스템에서 사용될 워크로드를 특정한 응용프로그램에 대해서만 제한하여 성능을 측정하지 않고, 가능한 모든 모델링 요소들에 대해서 변화를 주면서 테스트를 하여 Lustre 와 GPFS의 전반적인 성능을 비교하고자 하였고, 이를 통해서 WAN 클러스터 파일시스템에 적합한 것을 제안하려 하였다.

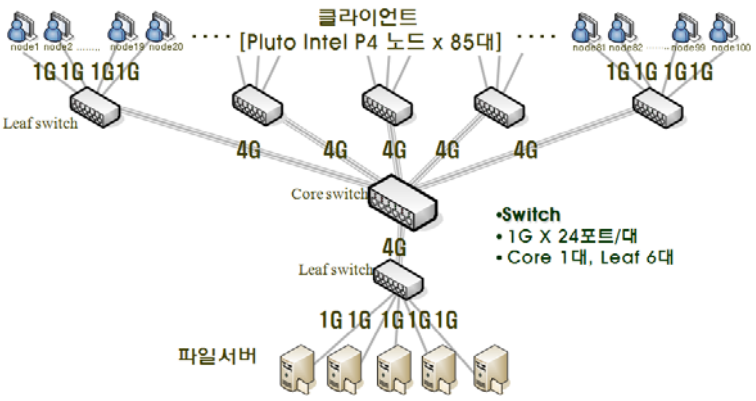
2. 클러스터 파일시스템에 따른 I/O 성능 분석

가. 테스트베드 구성

클러스터 파일시스템은 여러대의 서버와 클라이언트, 네트워크, 디스크 등 다양한 요소들이 하나로 모여 구성되므로, 각각의 환경구성은 매우 중요하다. 또한, Lustre와 GPFS 파일시스템의 소프트웨어적인 환경구성도 파일시스템의 I/O 성능이 큰 영향을 미친다. 다음은 하드웨어와 소프트웨어적인 환경구성을 보여준다.

(1) 하드웨어 구성

① 네트워크 구성



<그림 3-2> 네트워크 구성도

<그림 3-2>는 네트워크 구성을 보여준다. 구성될 클러스터 파일시스템은 그림처럼 스위치를 통하여 하나의 네트워크로 구성되어 있다. 종단 스위치에 연결된 각각의 서버와 클라이언트는 1 Gbps의 네트워크

전송량을 가지게 되며, 종단 스위치와 중앙 스위치 간에는 4 Gbps로 연결되어 있다. 따라서 각 클라이언트는 최대 네트워크성능을 1 Gbps 까지 가능하며, 5대의 서버가 제공할 수 있는 네트워크 성능은 중간의 4 Gbps의 병목구간으로 인하여 최대 4 Gbps 이다.

② 서버 / 클라이언트 테스트베드 구성

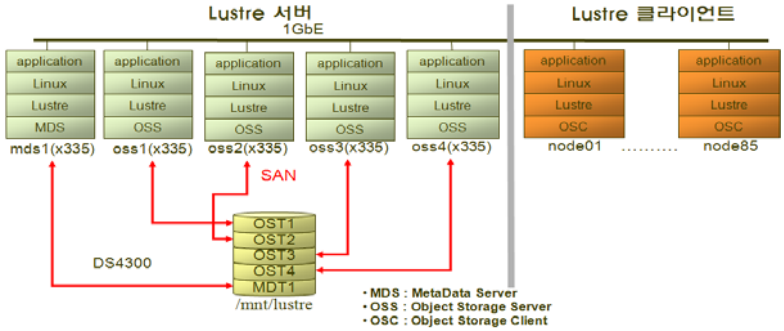
테스트에 사용된 서버와 클라이언트의 사양은 2장의 Lustre 환경구성과 동일하다

③ 디스크 구성

테스트에 사용된 디스크는 2장의 Lustre 환경구성과 동일하다.

(2) 소프트웨어 구성

① Lustre 파일시스템 구성



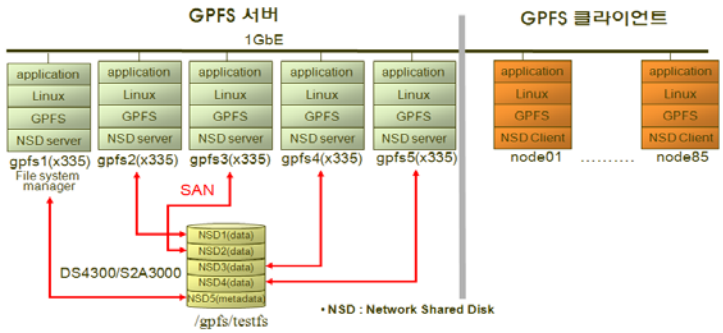
<그림 3-3> Lustre 클러스터 파일시스템 구성

테스트에 사용된 Lustre 클러스터 파일시스템은 DS4300 디스크를

4+1P Raid5레벨로 구성하여 4개의 파티션을 생성하였다. 4개의 파티션을 OST로 지정하여, 실제로 데이터 전송이 일어나는 OST 서버가 4대이고, DS4300의 나머지 남은 3개의 디스크를 묶어 MDT로 구성하여 메타데이터 연산을 처리할 수 있도록 하였다.

각각의 클라이언트는 MDS에 먼저 접근하여 메타데이터 연산을 한 후 OSS에 접근하여 실제 데이터 전송이 일어나게 된다.

② GPFS 파일시스템 구성



<그림 3-4> GPFS 클러스터 파일시스템 구성

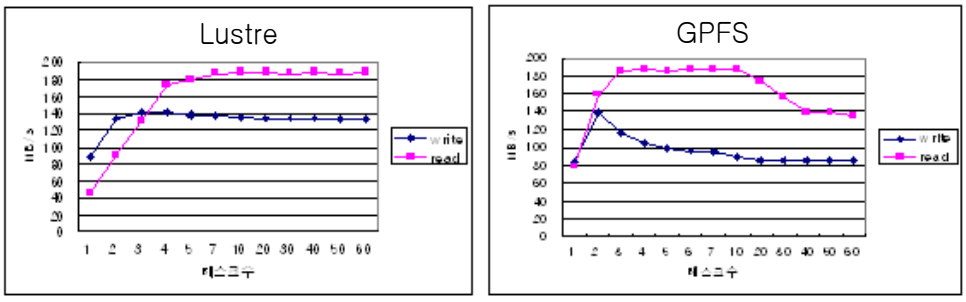
테스트에 사용된 GPFS 클러스터 파일시스템도 Lustre와 동일하게 DS4300디스크를 구성하여 4개의 데이터 서버와 1개의 메타데이터 서버를 구성하였다. GPFS는 메타데이터 서버를 따로 구성하지 않고, 전체가 다 메타데이터를 유지할 수 있도록 구성할 수 있지만, Lustre와의 비교를 위해서 메타데이터 전용 서버를 따로 구성하였다.

나. I/O 측정 결과

Lustre와 GPFS 클러스터 파일시스템의 I/O 성능을 분석하기 위해서 모델

링 가능한 다양한 요소들을 모두 변수로 두고 테스트하여, 특정한 워크로드가 아닌 파일시스템의 전체적인 성능을 비교하고자 하였다. 이에 따라 클러스터 파일시스템의 중요한 부분 중 하나인 확장성을 테스트하기 위한 태스크 수에 따른 I/O 성능을 분석하였고, 또한 I/O 액세스 패턴에 따른 성능, 블록크기에 따란 성능, 파일크기에 따른 성능을 분석하였다.

변수를 제외한 고정변수는 테스트베드의 환경에 따라 영향을 받지 않도록 잘 선택되어야 하는데 특히 파일크기는 I/O를 일으키는 클라이언트의 메모리 용량보다 커야 캐시효과를 제거할 수 있으므로, 정확한 I/O 성능을 측정하기 위해서 파일크기를 클라이언트의 메모리인 1 GB보다 2배 큰 2 GB로 설정해서 캐시 효과를 제거하였다. 블록크기는 파일시스템 블록크기 256 KB에 파일시스템 서버개수인 4배수로 하여 1 MB로 설정하였고 I/O 액세스 패턴에 대해서는 기본적으로 Sequential로 정하였다. I/O 액세스 모드는 파일 I/O에 참여하는 태스크가 증가할 때 하나의 파일을 모든 태스크가 액세스 하는지 또는 태스크마다 각자의 파일을 생성하여 액세스하는지를 나타내며, 기본적으로 File per process를 사용하였다.



<그림 3-5> 태스크 수에 따른 I/O 성능

(1) 태스크 수에 따른 I/O 성능

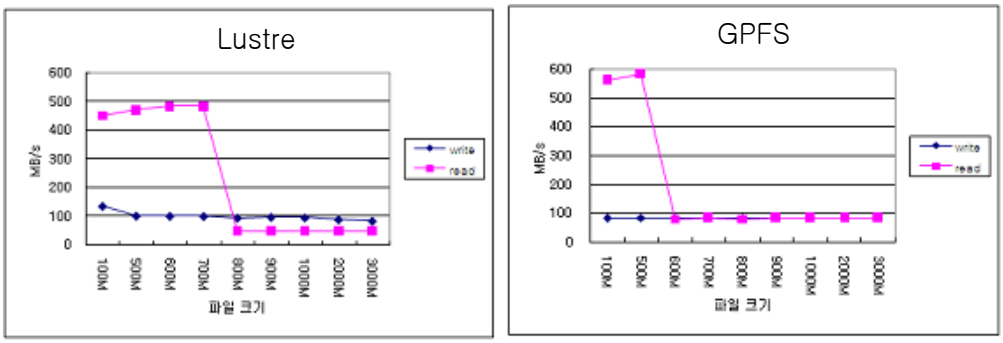
다음은 각 파일시스템의 확장성을 테스트하는 태스크 수에 따른 I/O 성능이다. 클러스터 파일시스템은 특성상 수많은 다른 클러스터들과 연결되어 I/O 서비스를 제공하기 때문에 높은 확장성을 제공하는 것은 가장 중요한 부분 중의 하나이다. 확장성을 테스트하기 위해서 각 클라이언트 노드 당 1개의 태스크를 생성하여 1~60개의 태스크까지 서버에 I/O를 요청하였다. 고정변수들은 다음과 같다.

- 파일크기 : 2 GB
- 블록크기 : 1 MB
- 액세스 모드 : File per Process
- 액세스 패턴 : Sequential

Lustre 에서는 read의 경우 태스크 수 4까지 write의 경우 태스크 수 2까지 I/O 성능이 올라가는 것을 볼 수 있고 GPFS의 경우에는 read와 write 모두 태스크 수 2까지 성능이 향상된다.

최대 성능은 write시 180MBps read시 140MBps로 Lustre와 GPFS 동일하며, GPFS가 Lustre보다 Read의 경우에 최대 성능에 더 빨리 도달하지만, 확장성의 측면에서는 Lustre가 태스크 수가 늘어남에도 성능의 저하 없이 더 좋은 성능을 보인다. 따라서 확장성이 가장 중요한 시스템에서는 Lustre가 GPFS보다 더 좋은 성능을 가질 것으로 보인다.

(2) 파일크기에 따른 I/O 성능



<그림 3-6> 파일크기에 따른 I/O 성능

위의 그래프는 파일크기에 따른 I/O 성능이다. 파일크기에 따라서 물리적인 디스크 I/O의 성능에 영향을 주지는 않지만, I/O를 발생시키는 클라이언트의 메모리 크기보다 작은 파일의 I/O를 할 때에는 클라이언트에 이미 캐시된 부분을 재사용함으로 인해서 성능이 몇 배 이상 향상된다. 하지만 이것은 디스크 자체의 I/O 성능이라 볼 수 없고, 파일시스템을 구성하는 디스크, 메모리, 운영체제의 상호작용에 따른 유기적인 성능이며, 가장 큰 영향을 주는 부분은 메모리의 크기이다.

고정변수들을 다음과 같이 설정하고 시험하였다.

- 블록크기 : 1 MB
- 액세스 모드 : File per Process
- 액세스 패턴 : Sequential
- 태스크 수 : 1

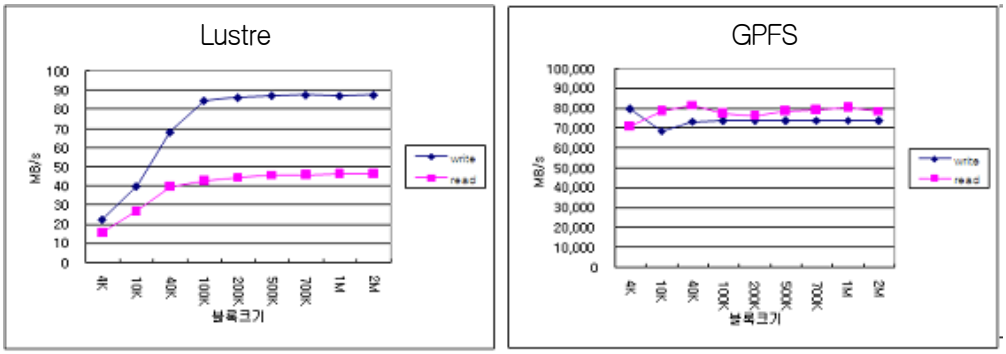
시험결과를 보면 Lustre와 GPFS 모두 일정한 파일크기 내에서는 read 성

능이 매우 좋아지는 구간이 있으며 그 이후에는 파일크기에 따른 성능의 변화는 거의 없다. Lustre의 경우에는 700 MB의 파일크기 까지 캐시 효과가 나타나고 GPFS의 경우에는 500 MB까지 캐시 효과가 나타난다. 이것에 대한 차이는 파일을 캐슁하는 방법에 의한 차이 때문인데, Lustre는 리눅스의 VFS 에서 사용하는 방법을 동일하게 사용하며, 시스템의 가용한 메모리 영역을 사용해 파일을 캐시한다. 반면에 GPFS 는 각 GPFS 클라이언트 마다 Pagepool 이라는 부분을 따로 설정하여 이 부분을 GPFS 파일캐시 전용으로 사용한다.

테스트에 사용된 클라이언트는 메모리가 1GB로 Lustre의 경우에는 OS에서 사용하고 있는 부분을 제외한 가용한 부분인 700MB 정도 캐시 효과가 나타나고, GPFS의 경우에는 Pagepool을 512M로 설정하여 500MB까지 캐시 효과가 나타나는 것을 볼 수 있다.

캐시를 사용함으로서 Write 연산을 바로 수행하지 않고 일정한 양을 모아 한 번에 처리하여 성능의 향상효과가 있으며, 특히 Random 한 패턴의 Write 연산의 경우에 특히 성능의 향상을 기대할 수 있다. Read 연산의 경우에도 필요할 것으로 예상되는 부분을 미리 읽어오는 Read-Ahead를 사용하거나, 사용했던 부분을 재사용하는 경우가 많을 때 성능의 향상을 기대 할 수 있다. 하지만 Lustre나 GPFS 같은 클러스터 파일시스템의 경우에는 각 클라이언트 마다 캐시의 내용을 동일하게 유지하는 것이 중요한 문제가 되며, 따라서 파일의 같은 부분을 여러 태스크가 동시에 사용할 때는 그 부분에 대한 캐시는 한 곳에만 있어야 하므로, 캐시를 충분히 사용하지 못할 가능성이 있다.

(3) 블록크기에 따른 I/O 성능



<그림 3-7> 블록크기에 따른 I/O 성능

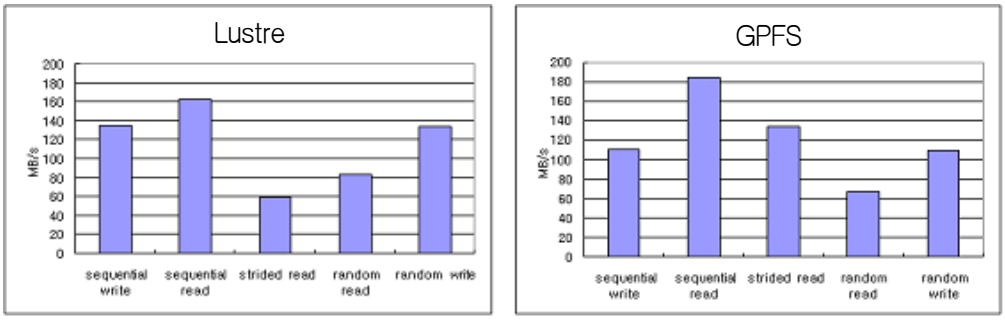
위의 그래프는 블록크기에 따른 I/O 성능이다. 블록의 크기가 작을수록 같은 크기의 파일에 대해서 더 많은 I/O 연산을 해야 하므로 성능이 저하될 가능성이 있다.

고정변수는 다음과 같다.

- 파일크기 : 2 GB
- 액세스 모드 : File per Process
- 액세스 패턴 : Sequential
- 태스크 수 : 1

시험결과 실제로 Lustre의 경우에는 블록크기가 감소함에 따라서 I/O 성능이 떨어지는 것을 볼 수 있으며, GPFS의 경우에는 블록크기에 크게 영향을 받지 않는 것을 볼 수 있다. 따라서 만약 사용되는 워크로드가 작은 블록크기의 I/O가 빈번하다면 GPFS를 사용하는 것이 좋을 것으로 예상된다.

(4) 액세스 패턴에 따른 I/O 성능



<그림 3-8> 액세스 패턴에 따른 I/O 성능

위의 그래프는 액세스 패턴에 따른 I/O 성능을 보여주는 그래프이다. I/O 액세스 패턴에 따라서 물리적인 움직임이 달라지고 이에 따라 디스크의 성능이 변한다.

고정변수는 다음과 같다.

- 파일크기 : 2 GB
- 블록크기 : 1MB
- 액세스 모드 : File per Process
- 태스크 수 : 1
- Strided Interval : 17 × 1MB(블록크기)

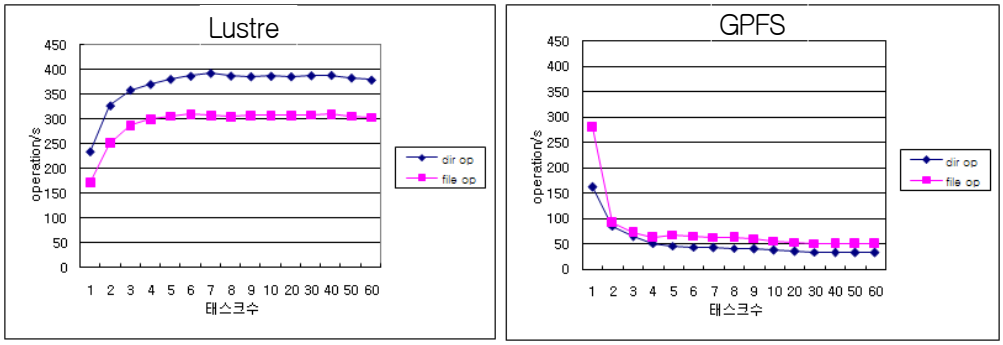
write의 경우에는 Lustre와 GPFS 공통적으로 Write-back의 영향으로 큰 성능차이를 나타내지 않는다. read의 경우에는 write 같이 작업을 뒤로 미뤘다가 수행할 수 없으므로 바로 수행이 된다. Sequential 패턴에 비해 Random과 Strided 패턴은 물리적인 디스크의 움직임이 많고, Read-ahead를 실패할 확률이 높으므로 낮은 성능을 보인다.

(5) 태스크 수에 따른 메타데이터의 성능

파일시스템의 성능에 있어서 메타데이터를 처리하는 성능은 매우 중요하다. 모든 I/O 연산을 위해서는 먼저 메타데이터 연산을 통해 액세스 하기 위한 파일에 대한 정보를 얻어 와야 하므로 메타데이터를 처리하는 성능이 떨어진다면 전체 파일시스템에 대한 병목구간이 될 수가 있다. 특히 클러스터 파일시스템은 I/O를 발생시키는 클라이언트가 많은 특징을 갖기 때문에, 동시에 얼마나 많은 메타데이터를 처리할 수 있는가는 클러스터 파일시스템의 확장성과도 직결된다.

Lustre와 GPFS의 메타데이터 처리 성능을 측정하기 위해서 두 부분을 측정하였는데, 첫 번째는 하나의 디렉터리 안에서 모든 태스크가 동시에 메타데이터 연산을 발생 시키고, 두 번째는 각각의 태스크마다 자신의 디렉터리를 가지고 그 안에서 메타데이터 연산을 발생시켰다. 이 두 가지의 차이는 create나 remove로 인해 발생하는 메타데이터 연산의 성능에 영향을 미치게 되는데, 이 두 연산은 시작하기 전에 자신을 포함하고 있는 디렉터리의 inode에 lock을 필요로 하므로 같은 디렉터리 안에서 모든 태스크가 동시에 메타데이터 연산을 발생시킬 경우 lock을 획득하기 위한 경쟁이 심해져 메타데이터 처리 성능이 저하된다.

① 공유 디렉터리



<그림 3-9> 메타데이터 성능(Shared directory)

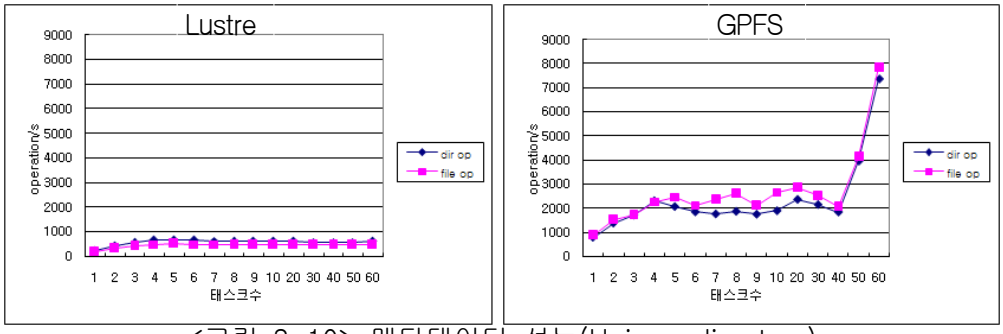
위의 그래프에서 한 번의 메타데이터 연산은 create, stat, remove 세 가지로 이루어지고, 1개부터 60개 사이의 태스크가 같은 디렉터리 안에서 메타데이터 연산을 동시에 각각 100번씩 발생시킨 결과이다. 결과를 보면 Lustre의 경우에는 점점 증가하여 디렉터리 메타데이터 연산은 400op/s 로 파일 메타데이터 연산은 300op/s 로 수렴하고, GPFS 같은 경우는 점점 감소하여 디렉터리 메타데이터 연산과 파일 메타데이터 연산 모두 50op/s로 Lustre에 미치지 못하는 성능을 보인다.

이는 Lustre가 Intent-based Locking 메커니즘을 사용하여, Look Up 및 Lock을 하기 위한 RPC 통신 횟수를 단 1번으로 줄여 시간을 단축시켰기 때문이다. Intent-based Locking 메커니즘은 Locking을 위한 Lookup을 수행할 때, Locking을 위한 Look Up 임을 지정하여 Look up에 성공 할 경우 자동으로 Lock을 할 수 있는지 결정하여 그에 대한 결과를 돌려주게 된다.

따라서 같은 디렉터리 안에서 파일과 디렉터리의 생성과 삭제가 빈번하게

일어나는 워크로드에 대해서는 Lustre 파일 시스템을 사용하는 것이 유리하다.

② 별도 디렉터리



<그림 3-10> 메타데이터 성능(Unique directory)

위의 그래프는 태스크 마다 100번의 메타데이터 연산을 각각의 디렉터리에 발생시킨 결과이다. 앞의 시험과 마찬가지로 한 번의 메타데이터 연산은 create, stat, remove 세 가지로 이루어지며, 차이점은 태스크마다 자신의 디렉터리를 가지고 그 안에서 메타데이터 연산을 수행한다는 점이다.

Lustre의 경우에는 공유 디렉터리의 경우와 비교해서 크게 차이가 나지 않지만, GPFS의 경우에는 큰 차이가 나서 8000op/s까지 증가하게 된다. 이는 GPFS 클라이언트의 메타데이터 캐시의 영향으로 인한 결과로 보이며, 얼마나 많은 메타데이터를 캐시하고 있을 것인지 클러스터의 환경으로 설정할 수 있다. 기본적으로 최대 캐시할 수 있는 파일개수의 4배의 크기로 설정이 된다. 공유 디렉터리처럼 같은 inode 정보를 공유하지 않기 때문에, 각각 클라이언트에 있는 캐시의 내용이 다른 클라이언트에 의해서 변경되지 않으므로 캐시의 효과가 나타나게 된다.

따라서 위와 같이 각각의 디렉터리를 가지고 메타데이터 연산이 많이 일어나는 경우에는 GPFS 파일시스템을 사용하는 것이 좋다.

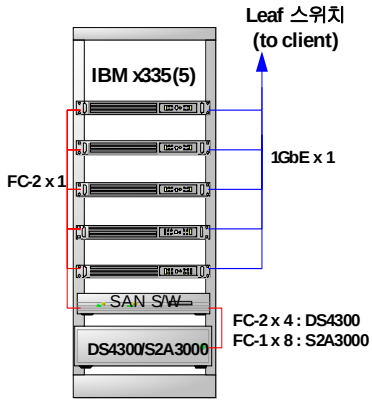
3. 디스크 저장장치에 따른 I/O 성능 분석 : DS4300 VS. S2A3000

가. 테스트베드

이번 절에는 디스크의 종류에 따른 성능의 변화를 보인다. 디스크마다 최대 전송률, 검색시간, 캐시용량, 전송률, 컨트롤러, 인터페이스 등의 구성이 다르므로 파일시스템의 전체적인 성능에 영향을 준다. <표 3-1>은 시험에 사용된 두 디스크를 보인다. 두 디스크는 SAN디스크로 SAN스위치를 통해서 GPFS 서버와 연결되어 있으며, <그림 3-11>이 이것을 보여준다.

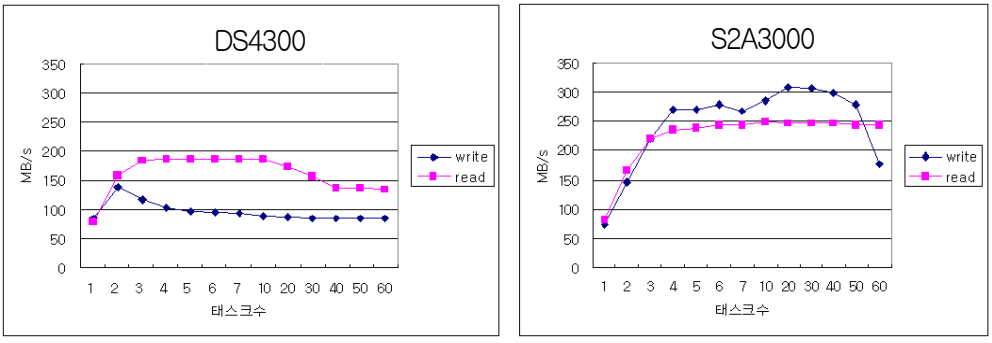
<표 3-1> 테스트베드 사양

IBM DS4300	DDN S2A3000
• Dual Controller • Host Interface : FC-2×4 • Cache : 512MB • Disk : FC Disk(134GB)×13 • Vendor Performance : 400MB/s	• Dual Controller • Host Interface : FC-1×8 • Cache : 3GB • Disk : FC Disk(134GB)×16 • Vendor Performance : 650~720MB/s



<그림 3-11> 디스크 구성도

나. I/O 성능 분석



<그림 3-12> 디스크에 따른 I/O 성능 분석

<그림 3-12>는 디스크에 따른 I/O 성능 시험결과이다. 다음과 같은 고정변수로 시험을 진행하였다.

- 파일크기 : 2 GB
- 블록크기 : 1MB

- 액세스 모드 : File per Process
- 액세스 패턴 : Sequential

전체적으로 S2A3000 디스크가 좋은 성능을 보이고 있으며 DS4300과 달리 read 와 write에 성능의 차이가 크게 나지 않는다. 또한 디스크의 성능에 따라서 파일시스템의 확장성에도 영향이 있는 것을 볼 수 있다. 디스크에 따라서 I/O 성능의 패턴과 확장성들이 다르므로, 사용할 워크로드에 따라서 적절한 디스크를 선택하는 것이 중요하다.

제4장 결론

원거리의 클러스터 간 데이터를 공유하기 위한 클러스터 파일시스템으로서 가장 많이 사용되는 Lustre 와 GPFS에 대해서 설명하였다. 각 Lustre 파일시스템을 구성하는 요소들과 이 구성요소들이 어떻게 유기적으로 연결되어 파일 I/O가 수행되는지를 보였다. 실제로 클러스터 파일시스템을 설치하기 위한 테스트베드 클러스터를 구성하여 GPFS와 Lustre를 클러스터 시스템에 설치하였다.

다음으로 클러스터 파일시스템 테스트베드에서 I/O 성능을 측정하였다. 태스크 수에 따른 I/O성능을 측정하여 클러스터 파일시스템의 확장성을 보였고, 파일크기, 블록크기, 액세스패턴에 대하여 성능을 측정하여 다양한 워크로드에 대해서 어떠한 I/O 성능을 보이는지 살펴보았다. 또한 확장성에 큰 영향을 미치는 메타데이터에 대한 처리성능을 시험하였으며, 사용하는 디스크 저장장치에 따라 어떠한 성능의 영향이 있는지에 대해 보였다.

참고문헌

1. Lustre datasheet & whitepaper

2. Object-based Cluster Storage Systems, Super Computing 2005, Brent Welch & David nagle, Panasas, Inc

3. Peter J.Braam, The Lustre Storage Architecture, Cluster File systems, Inc.

4. GPFS V 3.1 : Concept, Planning, and Installation Guide, IBM

5. Raymond L. Paden, GPFS Programming, Configuration and Performance Perspectives, GPFS Tutorial

6. Benny Mandler, Architectural and Design Issues in the General Parallel File System

7. IOZone User Guide

8. mdtest ReadMe file