그리드 환경에서 효율적인 작업 처리를 위한 대용량 파일 프로비저닝 방안

(Provisioning Scheme of Large Volume File for Efficient Job Execution in Grid Environment)

김 은 성 † 염 헌 영 ††
(Eunsung Kim) (Heon Y. Yeom)

요 약 그리드 환경에서 작업에 필요한 파일은 스테이징 기법에 의해서 전송된다. 이때 요구되는 파일이 대용량일 경우 전송에 걸리는 시간이 늘어나 작업의 시작 시간을 지연시키게 되다. 또한, 이는 작업의 전체 실행 시간을 증가시키는 요인이 되어서 작업처리량을 감소시키는 결과를 가져오게 된다. 따라서 이러한 파일 전송 시간 때문에 생기는 오버헤드를 줄인다면 그리드에서 실행되는 작업의 효율을 상당히 향상시킬 수 있다. 이러한 사실에 근거하여 본 논문에서는 그리드에서 효율적인 작업 처리를 위한 파일 프로비저닝 기법으로서 다음과 같은 두 가지 방법을 제안한다. 첫째, RA-RFT라는 방법을 제안한다. RA-RFT는 Globus Toolkit에서 파일 전송을 담당하는 RFT가 리플리카를 관리하는 RLS의 정보를 이용할 수 있도록 RFT를 확장한 것이다. RA-RFT는 파일 전송 시 가용한 리플리카들로부터 파일을 분할 전송함으로써 대용량 파일 전송 시간을 단축시킬 수 있다. 둘째, 리모트 링크라는 방법을 제안한다. 리모트 링크는 파일을 계산 노드로 직접 전송하지 않고 계산 노드에서 파일에 원격 접근할 수 있는 방법을 제공한다. 원격 접근 방법을 이용함으로써 계산 노드의 저장 공간을 절약할 수 있고 선반입을 통해서 효율적인 파일 프로비저닝을 가능하게 한다. 우리는 이러한 두 가지 방법이 기존 그리드 환경에서 사용하고 있는 스테이징 방식보다 우월한 성능을 보여준다는 것을 다양한 실험을 통해서 증명한다.

키워드 : 그리드, 파일 프로비저닝, 리플리카, 리모트 I/O, 스테이징

Abstract Staging technique is used to provide files for a job in the Grid. If a staged file has large volume, the start time of the job is delayed and the throughput of job in the Grid may decrease. Therefore, removal of staging overhead helps the Grid operate more efficiently. In this paper, we present two methods for efficient file provisioning to clear the overhead. First, we propose RA-RFT, which extends RFT of Globus Toolkit and enables it to utilize RLS with replica information. RA-RFT can reduce file transfer time by doing partial transfer for each replica in parallel. Second, we suggest Remote Link that uses remote I/O instead of file transfer. Remote link is able to save storage of computational nodes and enables fast file provisioning via prefetching. Through various experiments, we argue that our two methods have an advantage over existing staging techniques.

Key words : Grid, File provisioning, Replica, Remote I/O, Staging

† 학생회원 : 서울대학교 컴퓨터공학부
eskim@dcslab.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
yeom@dcslab.snu.ac.kr
논문접수 : 2009년 3월 20일
심사완료 : 2009년 6월 10일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제8호(2009.8)

1. 서 론

대부분의 과학 기술 응용 프로그램들은 복잡한 연산을 수행해야 되기 때문에 슈퍼컴퓨터나 클러스터와 같은 고성능 컴퓨팅 자원을 요구한다. 최근 들어 이러한 컴퓨팅 자원을 적은 비용으로 효율적으로 공급하기 위해서 그리드(Grid)와 같은 분산컴퓨팅 기술이 도입되고 있다. 그리드는 분산되어 있는 서로 다른 컴퓨팅 자원들을 네트워크로 연결하여 하나의 거대한 가상 컴퓨터처럼 사용할 수 있도록 해주는 기술이다[1]. 그리드를 구성하는 컴퓨팅 자원들의 통합적인 관리와 효과적인 사

용을 위하여 그리드 미들웨어가 사용되는데, Globus Toolkit(GT)[2]은 이러한 그리드의 개념을 실질적으로 구현한 대표적인 그리드 미들웨어이다.

GT는 효율적인 작업 실행과 데이터 관리에 필요한 다양한 서비스들을 제공하고 있다. GRAM(Grid Resource Allocation and Management)은 사용자가 요청한 작업을 실행하고 관리하는 기능을 제공하며 사용자가 작성한 작업 명세서에 기반하여 필요한 파일들을 준비하고 사용자 작업을 실행시킨다. 실행의 결과는 사용자에게 전송되거나 아니면 명세서에 지정된 위치로 전송된다. GT에서 파일 전송 서비스는 RFT(Reliable File Transfer)가 담당한다. RFT는 GridFTP 프로토콜을 이용하여 파일을 전송하며 파일 전송의 신뢰성을 보장할 수 있는 기능이 추가되어 있다. 따라서 파일이 전송되는 동안 주기적으로 파일 전송 상태가 데이터베이스에 저장되며 에러가 발생하더라도 에러가 발생하기 직전의 전송 상태로 복구할 수 있는 방법을 제공한다. 리플리카(replica)는 동일한 파일을 그리드 상의 다수 위치에 복제하여 생성한 파일을 의미한다. 리플리카는 파일에 대한 접근 지연 시간을 줄이고, 데이터 지역성을 높이며, 신뢰성을 보장해 준다. GT에서 이러한 리플리카들을 관리하기 위해 RLS(Replica Location Service)가 사용된다.

그리드 환경에서 작업에 필요한 파일들은 주로 스테이징(staging) 기법에 의해서 전송된다. 스테이징 기법은 작업이 할당된 계산노드로 요구되는 파일을 전송하는 것을 말한다. GT에서 파일의 스테이징은 GRAM이 작업 명세서에 지정된 파일을 RFT를 호출하여 계산 노드로 전송하는 것을 통해서 이루어진다. RFT에 의한 스테이징이 끝난 후 GRAM 서비스는 작업을 실행시킨다. 하지만 파일의 용량이 클 경우, 작업의 시작 시간이 지연되고 이는 전체 작업 처리량을 줄이는 결과를 초래할 수 있다. 본 논문에서는 이러한 파일 스테이징 시간을 줄이는 것에 초점을 맞춘다.

우선 파일 스테이징 시간을 줄이는 방법의 하나로 다수의 리플리카들로부터 같은 파일의 서로 다른 부분을 동시에 받아와서 파일의 전송 속도를 향상시키는 RA-RFT(Replica Aware-RFT)를 제시한다. RA-RFT는 리플리카 정보를 얻기 위해서 RLS를 이용하도록 RFT를 확장한 것이다. 즉, 같은 파일을 저장하고 있는 리플리카 목록을 RLS로부터 얻어서 각 리플리카로부터 동시에 서로 다른 부분을 전송하도록 한 것이다. 이 방법은 Gnutella와 같은 P2P 시스템[3]에서 사용하고 있는 방법을 응용한 것이다. 6개의 리플리카들을 이용하는 실험을 통해서, RA-RFT가 가장 좋은 성능을 갖는 리플리카 서버로부터 RFT를 이용하여 스테이징 하는 것 보다 약 4~5배의 성능 향상이 있다는 것을 보였다.

파일 스테이징 시간을 줄이는 또 다른 방법으로 파일을 직접 전송하는 대신 원격 I/O를 하도록 하는 리모트 링크(Remote Link) 방법을 제시한다. 리모트 링크는 작업 명세서에 지정된 원격 파일에 대한 링크 파일을 계산 노드에 생성한다. 본 논문에서는 이 링크 파일을 리모트 링크 파일(Remote Link File)이라고 칭한다. 원격 파일에 대한 리모트 링크 파일이 계산 노드에 생성되면 GRAM은 작업 명세서에 지정된 프로그램을 실행시킨다. 이 프로그램은 리모트 링크 파일을 통해서 원격 파일에 대한 I/O를 수행할 수 있다. 리모트 링크는 파일 시스템 관련 시스템 콜을 가로채서 I/O 서비스를 수행하는 리모트 링크 에이전트를 제공함으로써 원격 I/O를 위해서 사용자 프로그램의 수정을 요구하지 않는다. 그리고 파일 스테이징 시간을 없앨 수 있으며 계산 노드의 저장 공간을 전혀 이용하지 않는다는 장점을 제공한다. 실험 결과를 통해서, 선반입(prefetching)과 같은 최적화 기법이 적용된 리모트 링크가 RFT를 이용하여 파일을 전송하는 것 보다 약 2~3배 빨리 프로그램을 수행할 수 있음을 알 수 있었다.

본 논문에서 제시하는 두 가지 파일 프로비저닝 방법은 기존 그리드 환경에서 사용하고 있는 파일 스테이징 방법보다 더욱 효과적으로 작업에 요구되는 파일을 제공할 수 있고 이를 통해서 그리드의 작업 처리량을 향상시킬 수 있다.

이 논문의 나머지 부분은 다음과 같이 구성되어 있다. 2장에서는 복수의 리플리카를 활용하여 빠른 파일 전송을 가능하게 하는 RA-RFT 방법에 대해 살펴본다. 3장에서는 파일 전송 대신 리모트 I/O를 통해서 스테이징 오버헤드를 줄일 수 있는 리모트 링크 방법에 대해 설명한다. 4장에서는 RA-RFT와 리모트 링크의 성능 평가 결과를 제시하고 5장에서는 본 논문과 관련 있는 그리드에서 데이터 전송 시간을 줄이는 것에 대한 연구와 리모트 I/O 관련된 연구들에 대해서 알아본다. 마지막으로 6장에서 결론을 맺는다.

2. RA-RFT

본 장에서는 RFT와 RLS를 유기적으로 연동하여 파일 전송 효율을 높이는 RA-RFT의 시스템 구조와 다양한 부분 파일 전송 기법들에 대해 살펴본다.

2.1 시스템 구조

RLS는 그리드 상의 어떤 파일에 대한 다수의 리플리카들을 관리하기 위해서 LFN(Logical File Name)과 PFN(Physical File Name)을 사용한다. LFN은 그 파일을 유일하게 식별하는 논리적인 이름이고 PFN은 URI로 표현되는 그 파일의 위치나 주소이다. RLS에 의해서 관리되는 파일은 기본적으로 자신을 식별하는 LFN과

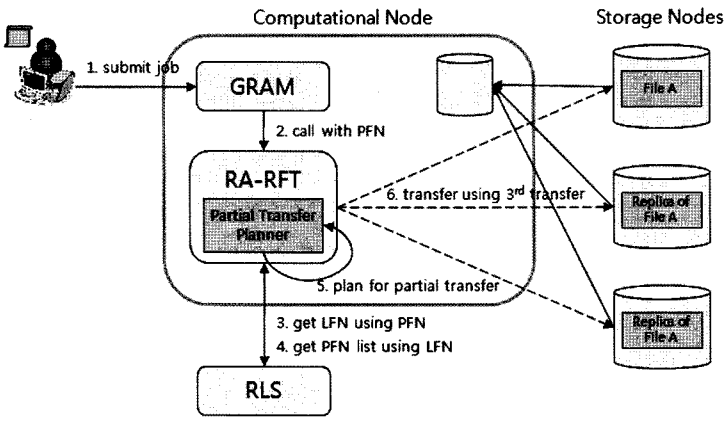


그림 1 RA-RFT 동작 과정

그리드 상의 실제 위치를 나타내는 PFN을 가진다. 만약 이 파일에 대한 리플리카가 생성되면 새로 생성된 위치를 나타내는 PFN이 추가된다. 따라서 파일은 단지 하나의 LFN만을 가지며, 하나 이상의 PFN들을 가질 수 있다.

RA-RFT의 동작 과정은 그림 1과 같다. 사용자 프로그램에서 이용하기 위해서 작업 명세서에 지정된 파일 이름은 PFN형태로 기술된다. 이 작업 명세서를 받은 GRAM은 파일을 스테이징 하기 위해서 이 PFN을 인자로 RA-RFT를 호출한다. RA-RFT는 우선 RLS로부터 이 PFN와 관련된 LFN을 구한다. 그리고 구해진 LFN을 사용해서 다시 PFN 리스트를 얻는다. 부분 전송 플래너(Partial Transfer Planner)는 이 PFN 리스트에 대해서 각 PFN으로부터 파일의 어떤 부분을 가져올 것인지를 관리자에 의해서 설정된 분할 정책에 따라서 전송 계획을 수립한다. 우리는 유니폼(uniform), 그리디(greedy), 프로브(probe) 기반 분할 방식을 제공한다. 마지막으로 RA-RFT는 설정된 분할 계획에 따라서 병렬로 각 리플리카들로부터 3rd 파티 전송을 통해서 계산노드로 파일을 스테이징 한다.

2.2 파일 분할 정책

효율적인 파일 프로비저닝을 위해 그림 2와 같이 유니폼 분할 방법, 그리디 분할 방법, 프로브 기반 분할 방법 등 세 가지 방법을 제시한다.

유니폼 분할 방법은 여러 개의 리플리카 서버들로부터 균등하게 파일의 일부분을 가져오는 방법이다. 이 방법은 각 리플리카 서버의 네트워크 처리량을 고려하지 않고 단순히 전체 파일 크기를 리플리카의 개수만큼 나누어서 전송한다. 예를 들어, 전체 파일 크기가 S이고 3개의 리플리카가 존재한다면, RA-RFT는 각 리플리카로부터 전체 파일의 S/3씩 받아 오게 된다. 하지만 유니폼 분할 방법에서는 네트워크 처리량이 가장 좋은 리플리카 서버와 가장 안 좋은 리플리카 서버가 같은 양의 데이터를 전송하게 된다. 따라서 전송 속도가 더 빠른 리플리카 서버는 느린 서버보다 자신이 맡은 데이터 전송을 더 빨리 끝내게 되고, 전체 파일 전송은 각 리플리카 서버에 할당된 부분이 모두 전송되어야 완료되는 것이므로, 전체 파일 전송 시간은 가장 느린 리플리카 서버의 파일 전송 시간에 달려 있다.

그리디 분할 방법에서 RA-RFT는 가지고 올 파일 부분을 슬라이스(slice) 단위로 각 리플리카 서버에 요청한다. 하나의 슬라이스 전송이 끝나면 다음 슬라이스를 받아오는 방식으로 동작하므로 파일 전송 속도가 빠른 리플리카 서버로부터는 많은 슬라이스들을 받아오게

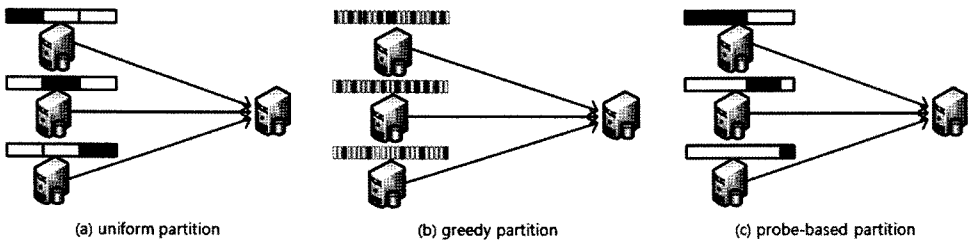


그림 2 파일 분할 정책

되고 그렇지 못한 서버로부터는 적은 슬라이스들을 받아오게 된다. 이는 서버의 성능과 네트워크 상태 변화에 따라서 가장 최적의 리플리카 서버로부터 더 많은 데이터를 전송 받을 수 있다는 장점이 있다. 그러나 이 방법은 슬라이스 크기가 작을 경우 각 리플리카 서버에 지나치게 많은 연결을 요구하게 되어서 서버의 전송 성능을 하락시킬 수가 있다. 그러므로 그리디 분할 방법에서는 슬라이스의 크기가 중요한 요소라고 할 수 있다. 우리는 다음 장에서 그리디 분할 방법에서의 슬라이스 크기에 대한 성능 변화에 대해 살펴본다.

프로브 기반 분할 방법은 각 리플리카 서버의 네트워크 성능을 고려하여 각 서버에 서로 다른 크기의 파일 일부분을 할당하여 빠른 파일 전송을 가능하게 한다. 네트워크 성능을 나타내는 지표는 지연시간, 처리량, 손실률 등 다양한 요소가 사용될 수 있다. 본 논문에서는 네트워크 처리량을 사용한다. 각 리플리카 서버의 네트워크 처리량을 구하기 위해서 NWS(Network Weather Service)[4]와 같은 대규모 네트워크 모니터링 시스템이 이용될 수 있지만 우리는 설치의 용이성을 고려하여 NWS 대신 netperf를 이용한다. Netperf[5]는 다양한 종류의 네트워크에 대한 성능을 측정할 수 있는 간단하지만 강력한 벤치마크 도구로서 양방향 네트워크 처리량과 종단 지연 시간 등을 측정할 수 있다.

다양한 실험 결과를 통해서 프로브 기반 분할 방법이 세가지 방법 중 가장 우수한 성능을 보임을 알 수 있었다.

2.3 신뢰성 있는 전송

분산 환경에서 오류는 불가피하게 발생할 수 밖에 없다. 그러므로 파일 전송 시 다양한 원인으로 인해 일어나는 오류들에 대처하기 위한 방법이 요구된다.

기존 RFT는 재시작 표시자(restart marker)의 형태로 파일 전송 상태를 데이터베이스에 기록하고 장애가 발생하였을 때, 익스포넨셜 백오프(exponential backoff) 알고리즘을 이용하여 재시작을 시도한다. 즉, GridFTP 장애, 네트워크 장애, 컨테이너 장애 등에 의해 파일 전송이 중단되면 RFT 서비스는 설정된 백오프 시간 후에 다시 전송을 시도하고, 또 다시 전송에 실패하면 배로 증가된 백오프 시간 동안 기다린 후에 파일 전송을 다시 시도한다.

RA-RFT는 여러 개의 리플리카들로부터 파일의 일부분을 동시에 전송하기 때문에, 장애로부터 전송 작업을 복구하기 위해서는 다수의 연결을 고려할 필요가 있다.

본 논문에서는 재귀 할당 기법을 소개한다. 재귀 할당 기법은 단순하면서도 여러 개의 오류가 발생해도 복구가 가능한 방법이다. 다수의 연결 중 하나가 오류가 발생하면 재귀 할당 기법은 오류가 발생한 리플리카 서버가 담당하고 있던 전송 부분을 오류가 발생하지 않고 동작중인 다른 리플리카 서버들에게 재분배한다. 재분배 정책은 유니폼 분할 방법에서는 임의로 선택하고 그리디 분할 방법에서는 오류가 발생한 리플리카 서버를 전송 대상에서 제거한다. 프로브 기반 분할 방법에서는 오류가 없는 서버들 중 가장 성능이 좋은 것으로 판단된 서버에 할당된다. 이렇게 새로 할당된 전송 부분은 기존에 할당 받은 전송이 다 끝나고 나서 시작된다. 따라서 재귀 할당 방법을 이용하면 오류가 발생한 리플리카 서버를 제외한 나머지 서버들로부터 파일 전송이 계속 진행될 수 있다. 이 방법을 통해서 RA-RFT 서비스는 다양한 오류 상황에서 신뢰성 있는 파일 전송 서비스를 제공할 수 있다.

3. 리모트 링크

본 장에서는 그리드 작업을 위해서 파일을 스테이징하는 대신에 리모트 I/O를 사용하여 작업 효율을 높일 수 있도록 하는 리모트 링크에 대해서 설명한다.

3.1 시스템 구조

심볼릭 링크(symbolic link)는 어떤 파일이나 디렉터리에 대한 참조를 포함하는 특수한 형태의 파일로서 POSIX를 지원하는 모든 운영체제에 의해서 지원된다. 심볼릭 링크는 실제 파일이나 디렉터리에 대한 경로만을 포함하므로 매우 적은 디스크 공간을 차지한다. 프로그램이 심볼릭 링크에 대해서 I/O 수행하면 이 I/O 요청은 실제 파일이나 디렉터리로 전송되어서 처리된다. 심볼릭 링크의 삭제는 실제 파일이나 디렉터리에 전혀 영향을 미치지 않는다.

본 논문에서는 이러한 심볼릭 링크의 개념을 차용하여 원격 파일에 대한 리모트 링크를 제안한다. 리모트 링크는 원격 파일에 대한 PFN을 포함하며, 프로그램에 의해서 사용될 때 로컬 I/O 요청을 PFN이 지정하는 노드에 대한 리모트 I/O로 전환하여 처리한다. 따라서 리모트 링크를 사용하는 프로그램은 원격에 있는 파일을 로컬에 있는 파일처럼 사용할 수 있다.

이러한 리모트 링크를 구현하기 위해서 사용자 레벨 프로그램에 파일시스템을 구현할 수 있도록 하는 FUSE (Filesystem in Userspace)[6]를 사용하였다. 리모트 I/O를 위해서 gsiftp 프로토콜을 기반으로 하는 GridFTP 클라이언트 라이브러리만을 이용하였지만, http나 ftp와 같은 프로토콜로도 쉽게 확장이 가능하다.

사용자가 리모트 링크를 이용할 수 있도록 GT 4.0.5 버전 이후로 지원되는 작업 명세서 확장 기능을 이용하여 <remoteLink> 엘리먼트를 새롭게 추가하였다. <remoteLink> 엘리먼트는 rlinkName과 srcFileUri와 같은 두 가지 속성값을 포함한다. rlinkName은 프로그램이 이용할 파일명과 동일한 이름을 가져야 하고 srcFileUri

```
<?xml version="1.0" encoding="UTF-8"?>
<job>
  <executable>/bin/cat</executable>
  <argument>${GLOBUS_USER_HOME}/.globus/rlinks/abc</argument>
  <extensions>
    <remoteLink name="rlinkName">abc</remoteLink>
    <remoteLink name="srcFileUri">gsiftp://sunset.snu.ac.kr/etc/hosts</remoteLink>
  </extensions>
</job>
```

그림 3 확장된 작업 명세서

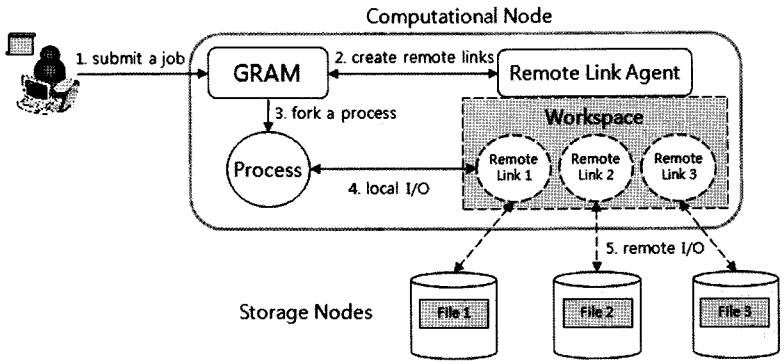


그림 4 리모트 링크 동작 과정

는 원격 파일에 대한 URI를 나타낸다. 그림 3은 <remoteLink> 엘리먼트를 사용한 작업 명세서 예를 보여준다. GRAM에서 이 엘리먼트를 인식하고 처리할 수 있도록 하기 위하여 Fork.pm과 JobManager.pm을 수정되었다. 사용자 프로그램을 실행시키기 전에 Fork.pm의 submit() 함수에서는 리모트 링크를 생성하고 프로그램이 종료한 후 JobManager.pm의 cache_cleanup() 함수에서는 리모트 링크를 삭제한다.

그림 4는 리모트 링크의 동작 과정을 나타낸다. 사용자는 파일 스테이징 대신에 리모트 링크를 이용하도록 <remoteLink> 엘리먼트를 사용하여 작업 명세서를 작성한다. 이 명세서를 받은 GRAM은 리모트 링크 에이전트에게 요구된 리모트 링크를 생성하도록 요청한다. 이 에이전트는 워크스페이스에 리모트 링크들을 생성한다. 워크스페이스의 경로는 설정가능하며 현재 구현에서는 ${GLOBUS_USER_HOME}/.globus/rlinks 디렉터리를 사용한다. 본 논문에서는 사용자가 미리 이 워크스페이스 경로를 알고 있어서 사용자 프로그램에서 이용할 파일의 경로가 이 디렉터리를 포함하고 있다고 가정한다. 리모트 링크 파일이 생성되면 GRAM은 프로그램을 실행시킨다. 프로그램은 워크스페이스에 있는 리모트 링크들에게 I/O 요청을 하고 리모트 링크는 이 요청을 PFN이 가리키는 노드로 리모트 I/O을 수행한다.

3.2 리모트 I/O 성능 최적화

프로그램이 파일의 데이터를 사용하는 방법은 순차 접근과 임의 접근으로 구분된다.

순차 접근의 경우 리모트 I/O의 장점은 데이터 전송과 계산이 중첩될 수 있다는 것이다. 즉, 원격 파일로부터 전송된 데이터를 프로그램에서 처리하는 동안 원격 파일의 또 다른 부분을 전송하는 것이 가능하다. 이를 지원하기 위해서 선반입 에이전트(prefetching agent)를 도입한다. 이 에이전트는 링 버퍼에 원격 파일로부터 가져온 데이터를 채우며 리모트 링크는 이 버퍼로부터 데이터를 가져와서 사용자 프로그램에 전달한다. 프로그램이 링 버퍼로부터 데이터를 비우는 속도보다 에이전트가 링 버퍼를 채우는 속도가 더 빠른 최적의 상황일 때, 프로그램이 요구하는 데이터는 항상 링 버퍼에 존재하므로 파일이 로컬에 존재하는 것보다 더 빠른 속도로 파일을 처리할 수 있다. 본 논문에서는 순차 접근 파일만을 다룬다.

임의 접근의 경우 다음에 가져올 데이터를 미리 예측할 수 없으므로 선반입 기법은 큰 도움이 되지 못한다. 임의 접근 파일에 대한 성능을 향상시키기 위해서 얻어온 데이터를 로컬 파일시스템에 저장하는 캐싱 기법이 사용될 수 있다. 이미 저장된 데이터에 대해서는 원격에서 다시 가져오지 않고 로컬에서 얻어올 수 있으므로 성능 향상에 도움을 줄 수 있다.

일반적으로 리모트 I/O의 성능은 네트워크와 파일 서

버의 상태에 전적으로 의존한다. 또한, 분산 환경에서의 리모트 I/O는 오류를 유발할 수 있다. 이러한 사항들를 고려하기 위해서 RA-RFT에서와 같이 RLS를 활용한다. 만약 데이터 전송 시간이 미리 설정된 임계값을 초과하게 되면 RLS로부터 그 원격 파일의 리플리카들의 정보를 얻어오고 netperf와 같은 툴을 사용해서 가장 성능이 좋은 리플리카의 위치를 찾는다. 선택된 리플리카의 PFN이 리모트 링크에 새롭게 등록되고, 이후 리모트 I/O는 이 리플리카에 대한 데이터 전송을 통해서 처리된다. 이와 같이 RLS를 이용함으로써 리모트 I/O의 성능과 신뢰성을 모두 보장할 수 있다.

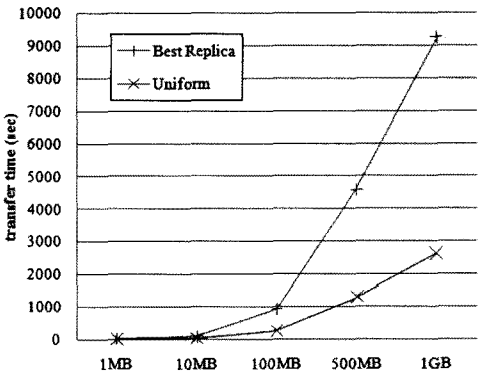
4. 성능 평가

본 장에서는 플래닛랩(PlanetLab)[7]에서 RA-RFT와 리모트 링크의 성능 측정 결과를 보인다. 플래닛랩은 범세계적 규모의 분산 시스템과 컴퓨터 네트워크 연구를 위한 테스트베드이다. 현재 플래닛랩은 전세계 460개 이상의 사이트에서 제공하는 880개 이상의 노드로 이루어져 있다. 본 논문의 실험 환경 구성은 다음과 같다. 클라이언트와 같은 로컬 네트워크 상의 한 노드는 GRAM, RFT와 RLS를 위해 구성되었고 또 다른 노드는 로컬 리플리카 서버로 구성되었다. 또한, 플래닛랩 상에 있는 6개의 노드들을 선정하여 원격 리플리카 서버로 구성하였다. 선정된 노드들은 Northeastern University(NU), Technique University Dresden(TUD), Indiana University(IU), Osaka City University(OCU), University of Minnesota(UoM), Chinese University of Hong Kong(CUoHK)이다.

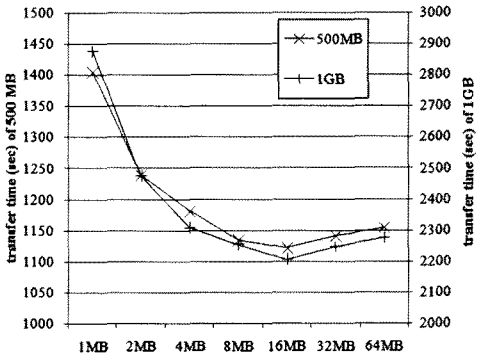
4.1 RA-RFT 실험

본 실험에서는 플래닛랩에 있는 6개의 원격 리플리카 서버들을 사용한다. 첫 번째 실험은 6개의 리플리카에 대해 유니폼 분할 정책을 사용하는 RA-RFT와 가장 성능이 좋은 리플리카 서버로부터 파일을 스테이징 하는 RFT의 평균 파일 전송 시간을 파일 사이즈를 변경하면서 비교 측정하였다. IU에 있는 플래닛랩 노드가 netperf를 통해서 구한 네트워크 처리량이 다른 노드들보다 큰 값을 보였기 때문에 가장 좋은 리플리카 서버로 선택되었다. 그림 5(a)는 이 실험 결과를 보여준다. 파일 크기가 커질수록 여러 개의 리플리카들을 활용하는 RA-RFT가 더 좋은 성능을 보여주는 것을 알 수 있다. 이는 유니폼 분할 정책이 본 논문에서 제시한 가장 원시적인 방법임에도 기존 스테이징 방법을 대체할 수 있는 효과적인 기법임을 증명하고 있다.

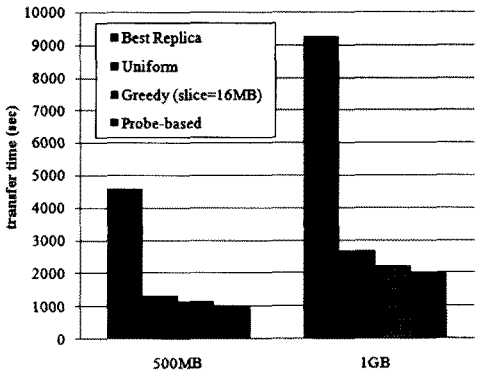
파일 슬라이스의 크기는 그리디 분할 정책의 성능을 결정하는데 가장 중요한 요소이다. 그림 5(b)는 그리디 분할 정책에서 슬라이스 크기의 영향을 보여준다. 파일



(a) 분할 전송 효과



(b) 슬라이스 크기의 영향



(c) 각 방식의 성능 비교

그림 5 RA-RFT 실험 결과

크기가 500MB와 1GB인 파일에 대해서 슬라이스 크기를 16MB로 나누는 것이 가장 좋은 성능을 나타내었다. 이 실험 결과를 근거로 다음 실험에 사용될 슬라이스 크기를 16MB로 정의하였다.

다음으로 다양한 분할 정책과 가장 좋은 리플리카를 활용한 RFT의 평균 전송 시간을 비교하였다. 그림 5(c)

는 각 방법의 평균 파일 전송 시간을 비교한 것이다. 이 실험을 통해서 프로브 기반 분할 정책이 그리드와 같은 분산 환경에서 파일을 프로비저닝하는 가장 효율적인 방법임을 알 수 있었다. 프로브 기반 정책은 가장 좋은 리플리카 서버로부터 파일을 전송하는 것 보다 약 4배 이상의 성능 향상을 나타내었다. 그리디 분할 정책은 많은 연결을 생성하는 오버헤드 때문에 프로브 기반 분할 정책보다 약간 성능이 떨어진다고 판단된다.

마지막으로 재귀 할당 방법의 실험을 위해서 가장 좋은 성능을 보여줬던 프로브 기반 분할 정책을 사용하였다. 6개의 리플리카 서버로부터 500MB 데이터를 전송하도록 하였고, 전송 중간에 몇 개의 리플리카 서버의 GridFTP 서비스를 강제적으로 종료하였다. GridFTP 서비스를 종료한 순서는 NU, TUD, UoM, 그리고 IU이다. 표 1은 이 실험을 수행한 결과이다. 이 결과는 다수의 서버에 오류가 발생하더라도 재귀 할당 방법에 의해서 파일 전송이 가능하다는 것을 보여준다. 실험 결과의 오버헤드는 오류가 없는 상태에서 전송에 걸린 시간에 대해서 오류가 발생했을 때 걸린 시간의 증가 비율을 의미한다. 3개와 4개 노드 장애 시 상대적으로 큰 오버헤드가 발생한 것은 네트워크 상태가 좋은 UoM와 IU의 GridFTP 서비스를 내렸기 때문이다. 그리드와 같은 대규모 분산환경에서는 장애가 잦기 때문에 본 논문에서 제시한 재귀 할당 방법은 상당히 중요한 기능이라고 할 수 있다.

표 1 재귀 할당 방법의 효과

Number of Failed nodes	Transfer time (sec)	Overhead (%)
0	1037.16	-
1	1379.12	32
2	1595.40	53
3	3253.18	213
4	5934.83	472

4.2 리모트 링크 실험

기존 그리드 환경에서 사용자는 작업에 필요한 파일을 스테이징 하기 위해서 다음 두 가지 방법을 이용하였다. 첫번째는 작업 명세서에 스테이징할 파일을 기술하여 GRAM이 RFT를 호출하여 스테이징하는 것이다. 두번째는 작업명세서에 기술하는 것이 아니라 작업으로 보낼 사용자 스크립트 내 globus-url-copy 명령어를 통해서 스테이징하는 것이다. 전자는 RFT에 의해서 신뢰성 있는 파일 전송이 가능하다는 장점이 있지만 신뢰성 보장을 위한 오버헤드로 파일 전송 시간이 길어지는 단점이 있다. 후자는 반대의 효과를 가진다. 리모트 링크의 효과를 증명하기 위해서 이 두 가지 방법과 비교를 수행하였다.

실험을 위해서 fb라는 바이너리 파일 뷰어를 사용한다. fb는 바이너리 파일에 대한 순차 접근을 한다. 본 실험에서는 fb가 사용자 작업으로 GRAM에 전송된 후, 각 기법에 의해서 요구되는 파일이 GRAM 노드에 준비되면 GRAM이 fb를 실행시켜서 바이너리 파일을 처음부터 끝까지 읽는 시간을 측정하였다.

실험에 사용된 파일은 500MB 파일이고, 이 파일은 로컬 리플리카 서버와 원격 리플리카 서버 중에서 CUoHK 노드 및 UoM 노드에 저장해 두었다. CUoHK는 UoM보다 더 큰 네트워크 처리량을 가진다.

그림 6은 500MB 파일에 대한 각 방식의 파일 전송 시간을 보여준다. 다른 크기의 파일에 대한 실험은 파일 크기에 따른 차이만 있을 뿐 비슷한 패턴을 보임으로 지면관계상 생략하였다.

RFT는 안정적인 파일 전송을 위해서 전송 상태에 대한 체크포인트를 수행하므로 가장 큰 전송 시간을 가진다. globus-url-copy는 작은 전송 시간을 가지지만 오류에 대처할 수 있는 방법을 제공하지 않는다. 리모트 링크는 globus-url-copy보다는 전송 시간에서 길지만 신뢰성 있는 파일 전송을 보장한다. 그리고 리모트 링크는 파일을 직접 전송할 때 생기는 저장 공간의 낭비도 없으므로 대용량 파일 사용이 빈번한 그리드 환경에서 의미있는 방법이 될 것이다. 또한 선반입을 적용한 리모트 링크는 globus-url-copy보다 더 좋은 성능을 보인다.

4.3 RA-RFT와 리모트 링크

마지막으로 본 논문에서 제시한 두 가지 방법의 성능을 비교해본다. RA-RFT의 성능은 리플리카 개수에 의존한다. 반면 리모트 링크는 선정된 하나의 리플리카 서버의 성능에 의존한다. 따라서 직접적인 비교는 어렵지만, 가용한 리플리카 개수에 따라서 어떠한 방식이 유리한지를 판단하기 위해서 다음 실험을 진행하였다.

클라이언트는 500MB 크기의 파일을 처리하는 fb 프로그램을 작업으로 제출한다. 작업에 필요한 500MB 파

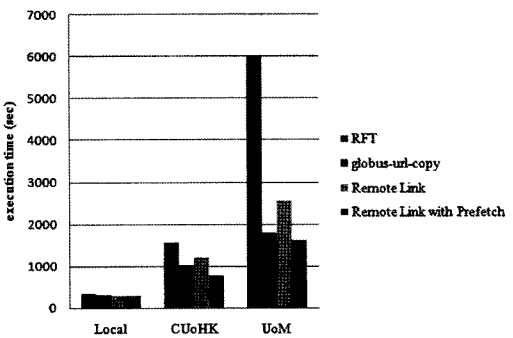


그림 6 리모트 링크 실험

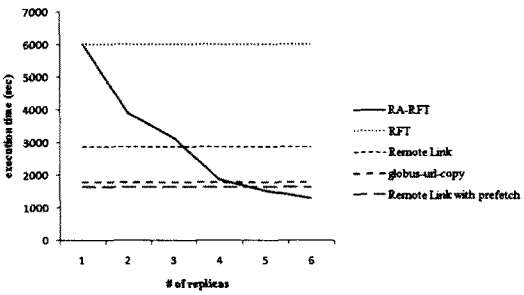


그림 7 RA-RFT와 리모트 링크 방법 비교

일은 각 기법에서 제시한 방법을 통해서 프로비저닝된다. 프로브 기반 분할 정책을 사용하는 RA-RFT는 리플리카의 개수를 1개부터 6개까지 늘려가면서 fb 실행 시간을 측정하였고, 리모트 링크와 RFT, globus-url-copy는 파일을 CUoHK 노드로부터 전송받을 때 fb 실행 시간을 측정하였다. 그림 7은 이 실험의 결과를 보여준다.

리모트 링크는 일정한 반면 RA-RFT는 리플리카의 개수가 많아질수록 리모트 링크보다 파일 전송 시간이 빨라짐을 알 수 있다. 상대적으로 네트워크 성능이 좋은 리플리카 서버가 추가되면 성능이 급격히 좋아지는 반면, 상대적으로 네트워크 성능이 좋지 않은 리플리카 서버가 추가되면 성능 향상이 적음을 그래프의 기울기를 통해서 알 수 있다.

우리는 이 실험을 통해서 리플리카 개수에 따라서 두 가지 방식이 상호 보완적인 관계임을 알 수 있다. 즉, 계산 노드의 저장 공간이 파일을 저장하기에 충분하고 가용한 리플리카의 개수가 충분하다면 RA-RFT를 사용하고 그렇지 않다면 리모트 링크를 선택하는 것이 바람직하다. 따라서 GRAM은 여유 저장 공간과 가용한 리플리카의 개수를 고려하여 두 가지 방식 중 하나를 적응적으로 선택해야 한다. 어떤 방식을 선택할 것인가를 결정하는 리플리카의 적절한 개수는 향후 연구 주제이다.

5. 관련 연구

그리드와 같은 글로벌 분산 환경에서 작업에 필요한 파일의 효율적인 프로비저닝에 대한 연구가 많이 진행되었다.

우선 리모트 링크와 같이 그리드에서 리모트 I/O를 수행하는 것에 대한 연구가 있다. GASS[8]는 GT에서 제공되는 컴포넌트로 리모트 I/O를 사용하여 프로그램이 원격에 있는 파일에 접근하도록 한다. 하지만 GASS는 자신의 API를 사용해서 사용자 프로그램을 수정할 것을 요구한다. RIO[9]와 RFS[10]는 그리드에서 실행되는 MPI 어플리케이션을 위한 것으로 다양한 이기종 환경에서 MPI 프로세스들이 리모트 I/O를 수행할 수 있도록 한다. 또한, GiSK[11]는 자신의 API를 이용하여 WSRF(Web Service Resource Framework) 기반 그리드 서비스들을 통해서 구성된 가상의 거대 저장소를 이용할 수 있도록 한다.

또 다른 연구로 그리드에 가상 파일 시스템을 구축하는 것에 대한 연구가 있다. SRM[12]은 대용량 스토리지 시스템을 그리드에서 접근 가능하게 하며, Gfarm[13]은 클러스터나 그리드에서 페타바이트 규모의 스토리지 구축을 위한 확장성 있는 I/O 대역폭과 병렬 처리를 제공한다. 이러한 가상 파일 시스템은 다양한 장점을 가지고 있지만 미리 계산 노드의 지정된 위치에 정적 마운트를 해 놓아야 되고 사용자가 미리 이 공유 디렉터리의 구조를 알고 있어야 한다.

그리드에서 리플리카를 활용하는 다양한 연구가 존재한다. DRS[14]는 새로운 리플리카를 생성하기 위해서 RLS와 RFT를 이용한다. 하지만 전송 속도를 향상시키는 방법은 제공하지 않는다. ReCon[15]은 RA-RFT와 같이 빠른 데이터 전송을 위해서 RLS를 사용한다. 가장 최적의 리플리카를 발견하기 위한 다양한 연구들이 [16-18]에서 진행되었다.

6. 결 론

그리드 환경의 작업 처리량을 향상시키기 위해서는 효율적으로 작업에서 요구하는 파일을 프로비저닝 할 필요가 있다. 이를 위해서 본 논문에서는 기존 그리드 환경에서 사용되고 있는 스테이징 기법을 개선하는 방법들을 제시하고 비교 분석하였다.

먼저 RA-RFT는 가용한 리플리카들을 활용하여 파일을 분할 전송함으로써 파일 전송 속도를 향상시킨다. 분할 정책으로서 유니폼 분할, 그리디 분할, 프로브 기반 분할 정책과 같은 다양한 방법들을 제시하였고 각 방법의 성능을 비교 분석하였다. 실험 결과를 통해서 네트워크의 현재 상태를 기반으로 하는 프로브 기반 분할 정책이 가장 효율적인 방법임을 알 수 있었다.

다음으로 직접적인 파일 전송 대신 리모트 I/O를 이용하는 리모트 링크를 제시하였다. 리모트 링크는 성능과 신뢰성을 보장하기 위해서 리플리카 정보를 활용하며 선반입 기법을 통해서 성능을 향상시킬 수 있었다. 실험 결과를 통해서 리모트 링크가 다양한 네트워크 상황에서 기존 방법들보다 우수한 성능을 보임을 알 수 있었다.

마지막으로 우리가 제시한 두 가지 방법을 비교하였다. 실험을 통해서 두 가지 방법은 상호 보완적인 관계임을 알 수 있었고 두 가지 방식 중 하나를 여유 저장 공간과 가용한 리플리카의 개수를 고려하여 적응적으로

선택해야 함을 알 수 있었다.

다음 연구로 우리는 리모트 링크에서 네트워크 상태에 따라서 적응적으로 선반입 블록의 크기를 변경하는 것을 진행하고 있다. 이것은 프로그램이 링 버퍼를 소비하는 속도에 기반하여 효율적으로 링 버퍼를 채우기 위해서 네트워크 전송 속도에 따라서 블록 크기를 변경하는 것을 포함한다. 또한 RA-RFT와 리모트 링크의 적응적인 선택을 위한 연구도 진행 중이다.

참 고 문 헌

[1] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Journal of Supercomputer Applications*, 15(3), 2001.

[2] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," *Conference on Network and Parallel Computing*, Tokyo, Japan, 2006.

[3] M. Ripeanu, I. Foster and A. Iamnitchi, "Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design," *IEEE Internet Computing*, 6(1), 2002.

[4] R. Wolski, N. T. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," Future Generation Computer Systems, 15 (5-6), 1999.

[5] Netperf, http://www.netperf.org.

[6] FUSE, http://fuse.sourceforge.net.

[7] PlanetLab, http://www.planet-lab.org.

[8] J. Bester, I. Foster, C. Kesselman, J. Tedesco, S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems," *Workshop on I/O in Parallel and Distributed Systems*, Atlanta, USA, 1999.

[9] I. Foster, D. Kohr, R. Krishnaiyer, J. Mogill, "Remote I/O: Fast Access to Distant Storage," *Workshop on Input/Output in Parallel and Distributed Systems*, San Jose, USA, 1997.

[10] Jonghyun Lee, R. Ross, R. Thakur, Xiaosong Ma, M. Winslett, "RFS: efficient and flexible remote file access for MPI-IO," *Conference on Cluster Computing*, San Diego, USA, 2004.

[11] Eunsung Kim, Hyeong S. Kim, Heon Y. Yeom, and Jongsook Lee, "GiSK: Making Secure, Reliable and Scalable VO Repository Virtualizing Generic Disks in the Grid," *Conference on High-Performance Computing in Asia-Pacific Region*, Beijing, China, 2005.

[12] A. Shishani, A. Sim, and J. Gu, "Storage Resource Managers: Middleware Components for Grid Storage," *Symposium on Mass Storage Systems*, College Park, USA, 2002.

[13] Osamu Tatebe, Noriyuki Soda, Youhei Morita, Satoshi Matsuoka, Satoshi Sekiguchi, "Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing," *Computing in High Energy and Nuclear Physics*, Interlaken, Switzerland, 2004.

[14] DRS, http://www.globus.org/toolkit/docs/4.0/techpreview/datarep/

[15] XiaoLi Zhou, Eunsung Kim, Jai Wug Kim, and Heon Y. Yeom, "ReCon: A Fast and Reliable Replica Retrieval Service for the Data Grid," *Symposium on Cluster Computing and the Grid*, Singapore, 2006.

[16] R. M. Rahman, K. Barker, and R. Alhajj, "Replica selection in grid environment: a data-mining approach," *Symposium on Applied computing*, Santa Fe, USA, 2005.

[17] D. Yin, B. Chen, and Y. Fang, "A fast replica selection algorithm for data grid," *Computer Software and Applications Conference*, Beijing, China, 2007.

[18] Y. Zhao and Y. Hu, "Gress-a grid replica selection service," *Parallel and Distributed Computing Systems*, Marina del Rey, USA, 2003.

김 은 성

2000년 성균관대학교 전기전자및컴퓨터공학부 학사. 2002년 성균관대학교 전기전자및컴퓨터공학과 석사. 2000년~현재 서울대학교 컴퓨터공학부 박사과정. 관심분야는 그리드컴퓨팅, 분산시스템, 멀티코어시스템, 운영체제

염 헌 영

1984년 서울대학교 계산통계학과 학사 1986년 Texas A&M Univ. 전산학 석사. 1992년 Texas A&M Univ. 전산학 박사. 1993년~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 분산시스템, 결함내성, 운영체제, 데이타베이스 시스템, 그리드 컴퓨팅