그리드 시스템에서 텍스트 파일 영역 관리
(Text File Region Management on Grids)

김 승 민 † 유 석 인 †† 김 일 곤 †††
(Seung Min Kim) (Suk I. Yoo) (Il Kon Kim)

요 약 프로그래밍 언어에서 변수를 사용하여 메모리의 특정 영역을 접근하는 것처럼, 텍스트 파일의 특정 영역도 문자열 이름을 가지는 변수로 표현하여 해당 영역의 값을 읽고 수정할 수 있는 텍스트 파일 인터페이스를 지원할 수 있다. 이것을 CAE, CAD, CAO 통합 자동화 시스템 분야에서는 파일 랩핑 (File Wrapping) 이라 한다. 파일 랩핑은 CAE, CAD, CAO 통합 자동화의 핵심 기술 중 하나이면서, 파일의 크기와 영역의 개수 및 분포에 따라서 통합 자동화 시스템의 실행 속도에 큰 영향을 미친다. 본 논문에서는 파일 랩핑의 핵심 기능을 일반화한 텍스트 파일 영역 관리 모델을 정의하고, 이를 그리드 서비스로 구현한 프로토타입의 구조를 설명한다. 그리고 구현된 프로토타입의 실행 결과 분석을 통하여, 제안된 모델과 텍스트 파일 영역 관리 서비스의 효용성을 검증한다.

키워드 : 그리드, CAE, CAD, CAO, 파일 랩핑, 파일 영역 관리

Abstract In the areas of CAE, CAD and CAO integration & automation technology, the word 'File Wrapping' means a virtualization of TEXT files that supports variables-based I/Os like variable assignments in programming languages. This File Wrapping process is one of the cornerstones of CAE, CAD and CAO integration & automation, and the performance of File Wrapping process, which is depending onthe size of a TEXT file to be accessed and the number of regions and their distribution, has a critical effect on the total performance of the CAE, CAD and CAO integration & automation systems. In this paper, we define TEXT File Region Management which generalizes the main functions of the File Wrapping process, and describe a prototype of TEXT File Region Management which is implemented as a Grid service. After that, the validity of the proposed model and the TEXT File Region Management service are discussed with evaluation results of the prototype.

Key words : GRID, CAE, CAD, CAO, File Wrapping, File Region Management

1. 서 론

그리드 컴퓨팅의 발전을 초기부터 지금까지 이끌고 있는 근본 동기는 컴퓨팅 자원들을 표준화된 IT 기술로 공유하는 것이다. 이 때, 컴퓨팅 자원들은 다양한 네트워크 구조로, 하나 이상의 서로 독립적인 운영 기관들에 분산될 수 있으며, 또한 시간의 변화에 따라 얼마든지 동적으로 연결 관계가 변경될 수가 있다.

이와 같은 특성을 가진 그리드에 대한 연구가 가시적인 성과를 보이면서, 이전의 슈퍼 컴퓨팅 및 클러스터 환경에서 운영되는 산업체 시스템들을 그리드 환경에서 운영할 수 있도록 시스템 구조를 변화하는 것에 대한 논의가 관련 학회와 업계에서 활발히 진행되고 있다. 특히 제조품의 단가가 상대적으로 높은 자동차, 조선 및 항공 등의 중기계 설계에 사용되는 CAE(Computer Aided Engineering), CAD(Computer Aided Design), CAO(Computer Aided Optimization)의 통합 전산 환경을 그리드 시스템으로 구축하는 것에 대해 관련 학회와 산업체에서 많은 관심을 가지고 있다[1,2].

중 기계 제조에서는 제품을 설계하는 과정이 적은 비용으로 빠른 시간 안에 진행되기를 원한다. 그러므로 경제적인 기준으로 보면, 설계의 최종 검증 수준에 이르기

† 정 회 원 : 서울대학교 전기컴퓨터공학부
mowgli@ailab.snu.ac.kr
†† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
siyoo@ailab.snu.ac.kr
††† 종신회원 : 경북대학교 전자전기컴퓨터공학부 교수
ikkim@knu.ac.kr
논문접수 : 2006년 3월 31일
심사완료 : 2007년 11월 13일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제13권 제7호(2007.12)

전까지는 제품을 실물로 만들어서 검증을 하는 것 보다 컴퓨터로 모델링 하여 이를 시뮬레이션 하여 검증하는 절차를 요구사항이 만족될 때까지 반복하면서 실험하는 것이 훨씬 유리하다. 따라서 이와 같은 시뮬레이션 기반 해석 실행을 조금이라도 빨리 끝내기 위해서는 고성능 컴퓨팅 파워를 가져다 줄 수 있는 슈퍼 컴퓨터와 클러스터 시스템과 같은 시스템들에 많은 관심을 가질 수 밖에 없으며, 같은 동기로 최근에는 그리드 컴퓨팅 시스템에 많은 관심을 가지고 있다.

CAE, CAD, CAO 통합 및 자동화 시스템을 구현하는 개발자들은 '파일 랩핑(File Wrapping)'이라는 용어를 흔히 사용한다. 이 용어가 의미하는 것은, 프로그래밍 언어에서 메모리를 변수들을 사용하여 접근하는 것처럼 텍스트 파일의 특정 영역들을 접근하는 것을 말한다(자세한 내용은 다음 장에 설명한다). 이 파일 랩핑 프로세스가 CAE, CAD, CAO 통합 및 자동화 방법론의 중요한 부분을 차지하고 있다. 따라서, 그리드 시스템에서 CAE, CAD, CAO 통합 및 자동화 환경을 구축하기 위해선 파일 랩핑 프로세스를 그리드 시스템에서 구현하는 것이 필요하다.

본 논문에서는 파일 랩핑의 핵심 기능인 텍스트 파일 영역 관리 기능을 그리드 환경에서 설계 및 구현한 내용을 기술한다. 또한, 구현된 프로토타입의 성능을 분석하여 CAE, CAD, CAO 통합 및 자동화 환경을 그리드 시스템으로 구성하였을 때 기대되는 유용성을 검토한다.

본 논문은 서론을 포함하여 5장으로 구성된다. 2장에서 텍스트 파일 영역 관리에 대하여 기술하며, 본 연구에서 제안하는 방법과 기존 소프트웨어에서 구현된 파일 영역 관리의 차별 점들을 기술한다. 3장에서는 본 연구에서 제시하는 텍스트 파일 영역 서비스의 구조에 대해 기술하며 4장에서 구현된 서비스의 성능 분석 결과를 기술한다. 5장에서 결론과 향후 과제를 정리한다.

2. 관련 연구

파일 랩핑은 텍스트 파일의 특정 영역을 문자열 이름을 가진 변수로 접근할 수 있는 인터페이스를 지원하는 기능이다. 본 장에서는 먼저, CAE, CAD, CAO 통합 및 자동화 환경에 파일 랩핑이 사용되는 구체적인 예를 통하여 파일 랩핑의 의미와 필요성을 살펴 본다.

예. 컴퓨터 시뮬레이션을 이용하여 두 자동차가 서로 정면 충돌을 하였을 때 발생하는 역학 현상을 실험한다. 이 때, 엔지니어는 A.exe 이라는 충돌 해석 시뮬레이션 프로그램을 이용해야 한다. A.exe 프로그램은 inputFile.txt 입력 파일을 사용하여 실행에 관련된 인자 값들과 옵션 등을 입력 받는다. 즉, inputFile.txt에 명시된 입력 인자 값들을 사용하여 성공적으로 A.exe 프로그램이 실

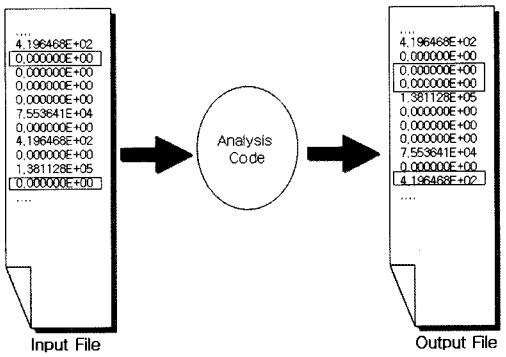

그림 1 텍스트 파일 입/출력 기반 해석 프로그램

행이 된 후에는 outputFile.txt 출력 파일에 시뮬레이션 결과가 저장된다(그림 1). 주어진 입력으로 충돌 해석을 한번 실행하는데 3~4시간이 소요된다.

특히 이번 실험에서는 엔지니어가 입력 정보 중 inputFile.txt 파일의 17라인(line)의 5번째에서 10번째 칼럼(column), 104라인의 52번째에서 73번째 칼럼 영역에 해당하는 값들을 변경해서 시뮬레이션을 실행해야 한다. 위 칼럼들에 대입할 값들은 1부터 100까지 정수로 총 100개이다. 실행 종료 후, 확인하고 싶은 내용은 outputFile.txt 파일의 1100번째 라인의 11번째에서 20번째 칼럼 영역에 저장이 되어 있다.

위 예에서 명시된 대로 시뮬레이션 기반 해석을 하기 위해서는 다음과 같은 4단계로 구성된 기본 절차들이 진행되어야 한다(그림 2).

1. 시뮬레이션 입력 값들이 저장되어야 할 텍스트 파일(들)의 특정 영역의 값을 새로운 값으로 변경한다.
2. 변경된 텍스트 파일(들)을 입력 파일로 설정하여 CAE 소프트웨어를 실행한다.
3. 실행이 정상적으로 종료 되면, 해석 결과가 저장된 텍스트 파일(들)이 생성된다.
4. 해석 결과가 출력된 파일(들)에서 특정 영역에 기록

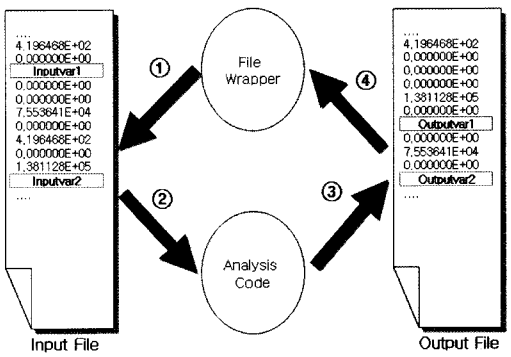

그림 2 파일 랩핑

된 값을 획득한다.

위 절차들을 실행하는 첫 번째 방법은 엔지니어가 매번 텍스트 파일 편집기를 사용하여 해당 영역의 값들을 원하는 값들로 수정한 후 프로그램을 실행하고, 그 결과를 역시 텍스트 편집기를 사용하여 확인하는 것이다. 두 번째는 다음과 유사한 형태의 프로그램 실행으로 위 작업을 자동화 할 수 있도록, 텍스트 파일 인터페이스를 구현하는 것이다.

```
Declare Begin
    InputVar1 inputFile.txt 17L 5:10C
    InputVar2 inputFile.txt 104L 52:73C
    OutputVar1 outputFile.txt 1100L 11:20C
Declare End
Program Begin
    LoopCounter = 1;
    For
        InputVar1 = LoopCounter
        Run "A.exe"
        Print OutputVar1
        LoopCounter++
  End For
Program End
```

일반적으로, 시뮬레이션 기반 해석을 구현한 대부분의 상용 CAE 소프트웨어(ANSYS, MSC NASTRAN, FLUENT 등)는 각각 공학 전문 분야의 체계적인 지식에 기반한 소프트웨어이다. 즉, 이 소프트웨어들은 유한 분석(Finite Element Analysis), 유체 해석(Computational Fluid Dynamics), 그리고 실험계획법(Design Of Experiments) 등과 같은 공학 해석 이론들을 바탕으로 구현한 소프트웨어이며, 각 소프트웨어마다 버전 업을 거치면서 해석 분야에 적합한 파일 포맷과 사용자 인터페이스를 가지고 있다. 따라서, 해당 소프트웨어에 대한 일정 기간의 교육을 받지 못한 엔지니어가 의미 있는 시뮬레이션을 실행하여 해석 결과를 얻는 것은 거의 불가능하다. 그럼에도 불구하고 다양한 공학 해석 분야를 동시에 고려하여 제조품의 제품설계를 진행해야만 하는 요구사항들이 계속해서 증가하고 있다.

두 개 이상의 공학 해석 분야 소프트웨어를 사용한 해석 작업들을 병행 진행하면서 특정 조건을 만족시키는 최적의 제품을 설계 - 이를 다분야 설계 (Multidisciplinary Design Optimization)라고 한다[3] - 하는 방법들로는 첫째, 엔지니어가 연동되어야 할 각각의 소프트웨어의 사용자 인터페이스를 직접 사용하여 입력 값들을 설정하고, 출력 값을 획득하는 방법과 둘째, 소프트웨어에서 제공하는 전용 API를 이용하여 입/출력값(들)을 관리하는 프로그램을 구현하여 실행하는 방법 그리고 텍스트 입/출력 파일(들)을 사용하여 소프트웨어가 실행되도록 설정 한 후, 파일 랩핑 기능을 사용하여 실행하는 방법들이 있다.

다분야 해석 자동화를 구현하기 위해서는 위 세가지 범주 가운데 첫 번째는 제외된다. 또한 두 번째 방법은 모든 시뮬레이션 기반 해석 프로그램들이 API를 제공하고 있지 않다. 이와 달리 파일 랩핑을 사용하는 마지막 방법은 텍스트 파일에 기록하기 때문에 당연히 발생하는 수치 자료 정밀성의 한계를 갖지만, 가장 범용적으로 사용되고 있다. 이는 지금까지 개발된 대부분의 공학 해석 프로그램들이 GUI 보다 텍스트 파일 입/출력을 더 선호해서 사용하는 FORTRAN으로 코딩 되었기 때문이다.

파일 랩핑은 여러 상용 PIDO(Process Integration & Design Optimization)와 MDO(Multi-Disciplinary Optimization) 소프트웨어에서 구현이 되어 있으며 대표적인 제품들로는 iSIGHT[4]와 ModelCenter[5] 그리고 FiPER[6]가 있다.

이들 소프트웨어 마다 구현 내용 및 용어에 약간의 차이가 존재한다. 그렇지만, 이들 다 하나의 파일에 대한 파일 랩핑 과정은 해당 파일을 접근할 수 있는 하나의 컴퓨터에서만 이루어진다는 것은 공통적인 사항이다. 따라서 파일 랩핑이 실행되는 컴퓨터의 메모리 용량을 크게 넘어서는 파일의 효율적인 처리에는 시스템 과부하가 발생한다.

일반적으로 시뮬레이션 공학 해석에 많은 시간이 소요될수록 해석 결과가 출력되는 파일(들)의 크기도 크다. 예를 들어 유체 해석의 경우, 유체를 mesh로 모델링을 해서 시뮬레이션 작업을 진행한다. 이때, 세밀한 해석을 위해서는 유체를 모델링 하는 mesh의 크기가 더 세밀해야 하므로, mesh의 개수가 많이 필요하게 된다. 따라서 각각의 mesh의 해석 결과를 기록하는 출력 파일의 크기도 당연히 증가하게 되어, 10G bytes 이상의 단일 파일 크기를 가지는 출력 파일들도 생성될 수 있다. 따라서 통합 해석의 실행 과정에서는 개별 해석 작업뿐만 아니라 해석간의 연동을 위한 파일 랩핑 작업에도 파일 크기가 커 질수록 PC나 UNIX, LINUX를 탑재한 단일 컴퓨터에서 작업을 진행하기에는 실행 시간이 많이 소요된다.

본 연구에서는 바로 이 문제점을 개선하기 위하여 그리드 환경에서 분산 파일 랩핑 기능을 시도하였다. 본 연구의 일차 목표는 파일 랩핑을 구현하는데 필요한 핵심적인 기능을 그리드 서비스로 지원하는 것이다. 그리고 이차적으로는 이 서비스를 이용하여 분산 파일 랩핑

을 구현하였을 때, 매우 큰 파일의 파일 랩핑에 소요되는 시간을, 기존 시스템에서 실행할 때 보다 얼마나 줄일 수 있는지를 분석하여 그 적용 가능성을 연구하는 것이다.

그리드 환경에서 특정 응용 분야에 제한을 두지 않은, 효율적인 데이타 관리를 위해서 많은 연구가 진행되고 있다. 이 가운데에서 본 연구에서는 GridFTP와 DAIS 작업 그룹(working group), 그리고 GFS 작업 그룹의 연구 내용들을 참조하였다.

GridFTP[7]는 인터넷 환경에서 데이타 전송 프로토콜로 가장 많이 사용되는 FTP 프로토콜의 그리드 버전이다. GridFTP는 그리드 환경에서 고성능으로, 안정적이며 보안을 고려한 그리드를 구성하는 컴퓨터들 사이의 대용량 데이타 전송 기능을 지원하기 위하여 구현되었다.

DAIS(Data Access and Integration Services) 작업 그룹은 GGF에 소속되어, 그리드 환경에서 데이타의 가상화를 실현하는 아키텍처를 구축하는 것을 목표로 하고 있다. 여기에 포함되는 핵심 서비스로는 데이타 찾기(discovery), 연합 접근(federated access), 작업흐름관리(workflow coordination), 동시성 관리(consistency management), 그리고 협업(collaboration) 지원 등이 있다[8].

GFS(Grid File System) 작업 그룹은 그리드 환경에서 일반 단일 컴퓨터에서 지원하는 파일 시스템과 유사한 형식으로 파일 서비스를 제공하기 위한 표준화 방안들에 대한 연구를 진행하고 있다[9]. 이 외에도 Andrew File System[10]과 NFS[11]등이 분산 환경에서의 저장장치 공유를 위한 대표적인 연구 결과물이다.

본 연구에서는 영역 관리를 적용할 파일 내용의 전송을 위하여, GridFTP을 사용한 데이타 전송도 블록 단위로 파일의 내용을 전송하는 기능의 대안으로 적용될 수 있도록 설계를 하였다. 또한, 동시성 관리 및 협업 기능 등을 대용량 파일 영역 관리에 지원하기 위하여 DAIS와 GFS 작업 그룹의 연구 결과들을 하부 구조로 활용하는 구조를 염두에 두었다.

3. 그리드 시스템에서의 텍스트 파일 영역 관리

본 장에서는 텍스트 파일 영역 관리를 정의하고, 이를 구현한 그리드 서비스의 구조에 대하여 기술한다.

3.1 정의

텍스트 파일 영역 관리의 정의를 위해서 텍스트 파일과 파일을 구성하는 라인들과 칼럼들을 다음과 같이 정의한다.

정의 1. 라인 l은 n개의 문자(들)로 구성된 문자열로 정의한다.

$$l = c_1c_2 \ldots c_n$$

이때, l^k는 c_k, $1 \le k \le n$를 표현한다.

정의 2. 텍스트 파일 f는 n개의 라인 l_k, $1 \le k \le n$ *(들)*을 원소로 가지는 전체 순서 집합(total ordered set)으로 정의한다.

$$f = <l_1, l_2, \ldots, l_n>$$

이때 f^k는 l_k, $1 \le k \le n$를 표현한다.

예를 들어, 첫번째 라인이 "abc", 두번째 라인이 "de fg"인 두 라인들로 구성된 파일 temp.txt는 위 정의에 따라 다음과 같이 표현된다.

$$temp.txt = < abc,\ de\ fg >.$$

위에서 정의한 라인 l 그리고 파일 f 에 적용되는 연산자들을 다음과 같이 정의한다.

정의 3. *NoOfLine(f)*는 텍스트 파일 f 의 라인 수를 획득하는 연산자이다. 따라서,

$f = <l_1, l_2, \ldots, l_n>$ 일 때, $NoOfLine(f) = n$

이다.

정의 4. *LengthOfLine(l)*는 라인 l을 구성하는 문자 개수를 획득하는 연산자이다. 따라서,

$l = c_1c_2 \ldots c_m$ 일 때, $LengthOfLine(l) = m$

이다.

정의 5. *ValueOfChar(f, n, m)*는 텍스트 파일 f의 n번째 라인의 m번째 문자를 획득하는 연산자이다. 따라서, $f=<l_1, l_2, \ldots, l_n>$ 일 때,

$$ValueOfChar(f, i, j) = (f^i)^j = (l_i)^j$$

이때, $1 \le i \le n$, $1 \le j \le LengthOfLine(l_i)$ 이다.

위의 정의들을 바탕으로 구간 $INT_f\ [A:B,\ C:D]$ 및 관련 연산자들을 다음과 같이 정의한다.

정의 6. $INT_f\ [A:B,\ C:D]$는 텍스트 파일 f의 A번째 라인의 B번째 문자부터 C번째 라인의 D번째 문자까지 구간을 표현한다. 이때, A, B, C, D는 다음 조건들을 만족해야 한다.

- $1 \le A \le NoOfLine(f)$
- $1 \le B \le LengthOfLine(f^A)$
- $A \le C \le NoOfLine(f)$
- $1 \le D \le LengthOfLine(f^C)$ *if* $A < C$
- $B < D \le LengthOfLine(f^C)$ *if* $A = C$

정의 7. Δ은 2개의 문자 c_1, c_2을 연결하여, 하나의 문자열로 만드는 연산자이다. 즉,

$$c_1 \Delta c_2 = c_1c_2$$

정의 8. *ValueOfInterval*$(INT_f\ [A:B,\ C:D])$은 구간 $INT_f\ [A:B,\ C:D]$에 저장된 문자열을 획득하는 연산자이다. 따라서, $f = <l_1, l_2, \ldots, l_n>$ 일 때:

$ValueOfInterval(INT_f\ [A:B,\ C:D])=$

$(ValueOfChar(f, A, B)\ \Delta ValueOfChar(f, A, B+1))$

$\Delta($ *ValueOfChar(f, A, B+2)* Δ*ValueOfChar(f, A, B+3))*

$\Delta\cdots$

Δ*(ValueOfChar(f, C, D-1)* Δ*ValueOfChar(f, C, D))*

이다.

위 정의들을 사용하여, 텍스트 파일 영역을 다음과 같이 정의한다.

정의 9. *영역(region) r*은 하나 이상의 구간 INT_f *[A:B, C:D]*들을 원소로 가진 집합이다.

$$r = \{INT_{f1}[A_1{:}B_1,\ C_1{:}D_1],\ INT_{f2}[A_2{:}B_2,\ C_2{:}D_2],\ \ldots,\ INT_{fn}[A_n{:}B_n,\ C_n{:}D_n]\}$$

정의 10. *ValueOfRegion(r)*은 영역 r에 저장된 문자열을 획득하는 연산자이다. 따라서,

$$ValueOfRegion(r)=\{ValueOfInterval(INT_{fi}[A_i{:}B_i, C_i{:}D_i]) \mid INT_{fi}\ [A_i{:}B_i,\ C_i{:}D_i] \in r\}$$

이다.

본 논문에서는 영역 r의 생성 및 삭제, *ValueOfRegion(r)*의 결과값 획득 및 변경 기능을 사용한 텍스트 파일 영역 접근을 텍스트 파일 영역 관리로 정의한다.

3.2 텍스트 파일 영역 관리 그리드 서비스 구조

본 연구에서는 위 절에서 정의한 텍스트 파일 영역 관리 기능을, 그리드 서비스 규정[12]을 준수하는 서비스(File Region Management Service, 줄여서 FRS)로 설계 및 구현하였다. FRS 서비스는 표 1과 같이 4개의 메쏘드로 구성되었다.

먼저, 영역 생성을 위해서는 특정 영역 r을 표현하는 symbol과 r을 구성하는 하나 이상의 구간 정보 INT_f *[A:B, C:D]*가 *defineRegion(...)* 메쏘드에 인자로 전달된다. 이렇게 생성된 영역 r의 값을 얻어오기 위해서는 *getRegion(...)* 메쏘드에 r의 symbol을 인자로 전달하며, 그 실행 결과로 r의 값 *ValueOfRegion(r)*이 획득된다. 영역 r의 값을 변경하기 위해서는 r의 symbol과 변경을 원하는 값이 *putRegion(...)*에 인자로 전달되어 호출된다. 그러면, 영역 r을 구성하는 각 구간의 *ValueOfInterval(*INT_f *[A:B, C:D])*의 값이 변경되도록 파일 내용이 char 단위로 수정된다. 마지막으로, *undefineRegion(...)*에 영역 r의 symbol 값을 전달하면 r에 대한 영역 관리를 중지한다.

표 1 FRS를 구성하는 메쏘드들

Name	Description
defineRegion(…)	영역을 생성한다
undefineRegion(…)	생성된 영역을 삭제한다
getRegion(…)	영역의 값을 획득한다
putRegion(…)	인자로 주어진 값으로 영역의 값을 변경한다

표 2 SDS를 구성하는 메쏘드들

Name	Description
getCapacity(…)	저장 공간을 기증할 컴퓨터의 정보를 획득한다
create(…)	기증한 저장 공간에서 파일을 생성한다
open(…)	기증한 저장 공간에서 파일을 오픈한다
delete(…)	기증한 저장 공간에서 파일을 삭제한다
close(…)	기증한 저장 공간에서 파일을 닫는다
seek(…)	기증한 저장 공간에 있는 파일의 파일 포인터 위치를 변경한다
send(…)	기증한 저장 공간에 있는 파일에 내용을 추가한다
recv(…)	기증한 저장 공간에 있는 파일의 내용을 획득한다
getFileInfo(…)	기증한 저장 공간에 있는 파일 정보를 획득한다
getFirstLine(…)	기증한 저장 공간에 있는 파일의 첫 번째 라인을 획득한다
getLastLine(…)	기증한 저장 공간에 있는 파일의 마지막 라인을 획득한다
getLine(…)	기증한 저장 공간에 있는 파일의 임의 라인을 획득한다

표 3 VSS를 구성하는 메쏘드들

Name	Description
addSD(…)	저장 공간을 기증할 컴퓨터를 등록한다
updateSD(…)	등록된 컴퓨터의 정보를 변경한다
removeSD(…)	저장 공간 기증 컴퓨터 목록에서 삭제한다

또한, 하나의 큰 파일에 대한 영역 관리를 그리드를 구성하는 각각의 컴퓨터들에서 분산하여 영역 관리를 하기 위해서, 부분 파일 전송 기능이 필요하였다. 이를 위해 저장소 기증 서비스(Storage Donor Service, 줄여서 SDS)와 가상 저장소 서비스(Virtual Storage Service, 줄여서 VSS)를 추가로 설계하였다(표 2).

가상 저장소 서비스는 저장소 기증 서비스를 구성하는 메쏘드들에 다음의 메쏘드들을 추가하여 설계하였다(표 3).

FRS, SDS, SHS 서비스 전체 서비스 연결 구조는 그림 3과 같다.

파일 영역 관리를 위해선 먼저 파일을 가상 저장소로 업로드(upload)한다. 그 후 파일 영역 관리를 클라이언트의 요청에 따라 진행하며, 파일 다운로드(download)가 실행되면 파일 영역 관리를 마치게 된다. 파일 업로딩, 영역 관리 그리고 다운로딩 별 세부 운영 절차는 다음과 같다.

파일 업로딩 절차

1. 클라이언트가 VSS에 영역 관리를 적용할 파일 업로딩을 요청한다.
2. VSS는 요청 받은 파일을 분산하여 저장할 수 있는

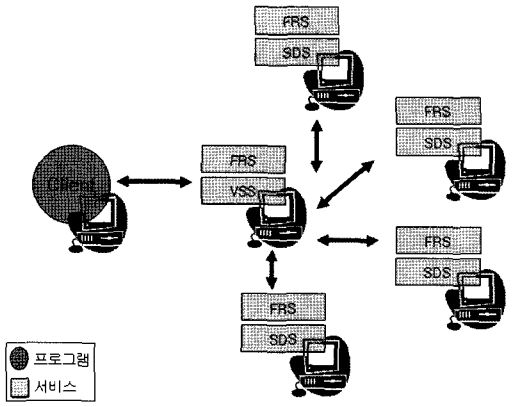

그림 3 서비스 구조

SDS들을 선택한 후, 개별 SDS마다 사용할 저장 공간을 확정한다.

3. 클라이언트가 파일 내용을 VSS에 전달하면, VSS는 단계 2에서 결정된 개별 SDS에게 파일 내용을 분산하여 전송한다.

파일 영역 관리 절차

1. 클라이언트가 VSS에 전송된 파일에 대한 파일 영역 서비스를 요청한다.
2. VSS는 자신과 연계되는 FRS를 클라이언트에 할당한다.
3. 클라이언트가 단계 2에서 획득한 FRS를 이용하여 파일 영역 생성, 값 획득 및 변경 그리고 삭제를 요청한다.
4. FRS는 단계 3에서 요청 받은 영역 정보를 개별 SDS 마다 저장된 부분 파일의 영역 정보로 변환하여, 영역이 존재하는 SDS(들)을 찾는다.
5. 단계 4에서 검색된 SDS(들)와 연계된 FRS(들)에게 변환된 영역 정보에 대한 파일 영역 생성, 값 획득 및 변경 그리고 삭제를 요청한다.
6. SDS와 연계된 FRS는 요청 받은 작업을 수행 후, 그 결과를 VSS와 연계된 FRS에 전달한다.
7. VSS와 연계된 FRS는 단계 6에서 획득한 부분 파일별 영역 정보를 전체 파일에 대한 정보로 변경 후, 클라이언트에게 전달한다.

파일 다운로딩 절차

1. 클라이언트가 VSS에 영역 관리가 적용되어 내용이 변경된 파일의 다운로딩을 요청한다.
2. VSS는 요청 받은 파일이 분산되어 저장된 SDS들을 검색한다.
3. 클라이언트가 변경된 파일 내용 획득을 VSS에 요청하면, VSS는 단계 2에서 결정된 개별 SDS에서 파일 내용을 획득하여 클라이언트에게 전송한다.

4. 성능 평가

본 장에서는 구현된 텍스트 파일 영역 관리서비스의 성능 평가 방법과 그 결과에 대해서 기술한다.

4.1 평가 방법

다음과 같은 6단계로 평가 절차를 구성하였다.

1. 파일 영역 관리 서비스를 적용할 텍스트 파일을 그리드를 구성하는 각각의 컴퓨터들에 전송한다.
2. defineRegion(…) 메쏘드를 호출하여 n개의 구간으로 구성된 영역을 생성한다.
3. getRegion(…) 메쏘드를 호출하여 생성된 영역의 값을 획득한다.
4. putRegion(…)메쏘드를 호출하여 외부에서 입력된 값으로 영역의 값을 변경한다.
5. undefineRegion(…) 메쏘드를 호출하여, 이미 생성된 영역을 삭제한다.
6. 그리드를 구성하는 각각의 컴퓨터에 전송된 부분 파일들을 다시 획득한 후 원본 파일과 같은 구조로 재구성한다.

평가의 주 목적은 구현된 그리드 서비스의 성능 평가이므로, 먼저 동일한 파일에 대하여 단일 컴퓨터에서 실행 한 후(위 평가 절차에서 1번과 6번을 제외한 실행), 이 결과를 1번부터 6번까지의 전체 절차를 실행한 그리드 서비스의 성능과 비교하였다.

4.2 프로토타입 및 테스트 환경

본 연구에서는 앞 장에서 살펴 본 FRS와 SDS 그리고 VSS의 프로토타입 서비스들을 Globus[13] Toolkit V3(GT 3.2.1)와 Java JDK 1.4.2_08을 사용하여 구현하였다. 또한, Intel Pentium4 2~3GHz CPU와 1~2Gbtyes의 RAM으로 구성된 개인용 컴퓨터 4대에, LINUX 계열의 OS를 설치하여 Ethernet 네트워크로 연결하였다(이들 컴퓨터들은 모두 하나의 스위치 허브를 통해 직접 연결되어 있다).

4.3 테스트 파일 크기 및 영역 개수

파일 영역 관리를 적용할 파일들은 파일 크기를 2M, 20M, 200M, 2G, 4G, 그리고 8G로 다양화 하였다. 그러나, 각각의 파일은 10 bytes로 구성된 라인들로 동일한 구조로 구성되었다. 따라서, 2Mbytes 크기의 파일은 200K의 라인들로 구성되며, 2Gbytes 크기의 파일은 200M의 라인들로 구성된다.

영역을 구성하는 구간들은 JDK 1.4에서 지원되는 *java.util.Random.nextInt(int)* 메쏘드를 사용하여 생성하였으며, 구간들의 위치는 테스트 파일의 전구간에 균등하게 분포되었음을 분석 프로그램을 사용하여 확인하였다.

4.4 실행 결과 및 분석

10Mbps의 네트워크 속도에서 테스트를 실행하여 그

림 4, 5, 6, 7, 8, 9에서 보여지는 결과를 얻었다.

먼저 구간 개수 변화에 따른 성능을 비교해 보자. 그림 4, 5, 6, 7은 각각 2M, 20M, 200M, 2G의 파일에서 구간 개수를 1, 5, 10,50, 100, 500, 1000으로 증가하면서 파일 영역 관리를 실행한 속도를 측정한 결과를 보여 준다. 구체적인 수치는 다르지만 이들 모두 공통된 패턴을 보이는 것을 확인할 수 있다. 즉, 테스트를 진행한 모든 파일들에서 구간 개수 1개에서는 그리드 시스템의 실행 소요 시간이 단일 시스템보다 더 소요된다. 그런데, 해당 파일 내에 구간 개수가 10개를 넘게 되면, 단일 시스템 보다 그리드 시스템에서 실행 소요 시간이 더 짧아지며, 구간 개수가 증가할수록 단일 시스템과의 간격이 더 벌어지는 것을 볼 수 있다.

이와 같은 결과가 발생하는 이유는 다음과 같다. 그리드 시스템을 구성하는 컴퓨터가 n 개가 있다고 가정하자. 그리드 시스템에서 파일 영역 관리에 소요되는 전체 시간 T_{total}는 크게, 그리드를 구성하는 모든 컴퓨터에 전체 파일의 부분 내용을 전송하는데 소요되는 시간 T_{upload}와 이들 각각의 컴퓨터에서 파일 영역 관리에 소요되는 시간 $T_i (1 \le i \le n)$의 최대값, 그리고 수정된 부분 내용들을 개별 컴퓨터에서 전송 받아 하나의 파일로 만드는 데 소요되는 시간 $T_{download}$의 합으로 구성된다.

$$T_{total} = T_{upload} + MAX(T_1, T_2, \ldots, T_n) + T_{download}$$

즉, 그리드 시스템에서 실행되는 모든 파일 영역 관리의 실행 시간에는 구간 개수의 많고 적음에 상관없이 T_{upload}와 $T_{download}$가 고정으로 포함 된다. 따라서, 단일 시스템에서의 파일 영역 관리에 소요되는 시간이 ($T_{upload} + T_{download}$) 보다 적을 경우는 본 방법을 적용하는 것이 비효율적이다. 그러나, 위 실험 결과에서도 보여지듯이, 파일의 구간 개수가 많아질수록, 또한 그리드 시스템에서 사용할 수 있는 컴퓨터의 개수가 증가할수록 그리드 시스템에서 파일 영역 관리를 실행하는 것이 단일 시스템에서 실행하는 것 보다 더 유리하다.

구간 개수를 고정하고, 파일 크기를 변화하면서 성능 비교를 해 보면 그리드 시스템에서의 파일 영역 관리와 단일 시스템에서의 파일 영역 관리의 차이점을 더욱 뚜렷하게 볼 수 있다. 그림 8, 9에서는 구간 개수 100개와 1,000개의 두 가지 경우에 대해서, 파일 크기를 2M, 20M, 200M, 2G로 변화함에 따라 단일 시스템과 그리드 시스템에서의 실행 소요 시간을 비교하였다. 파일의 크기가 커질수록 단일 시스템과 그리드 시스템 모두 파일 영역 관리에 시간이 증가하는 것은 동일하다. 그러

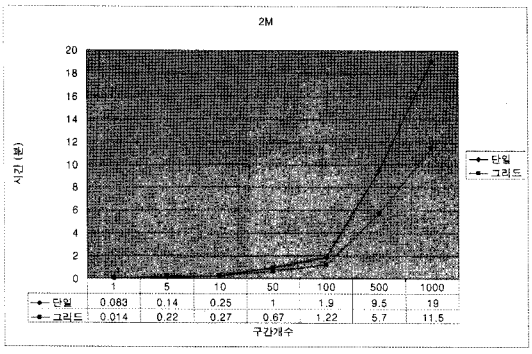

그림 4 2M 파일에서 실행 결과 (10Mbps)

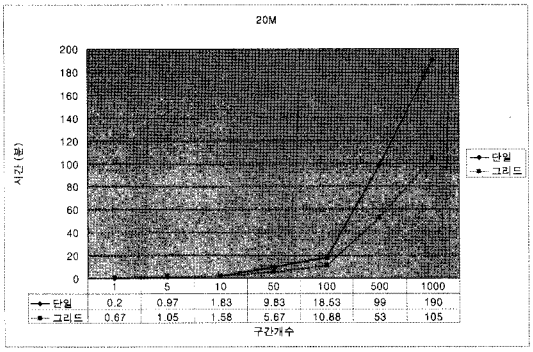

그림 5 20M 파일에서 실행 결과 (10Mbps)

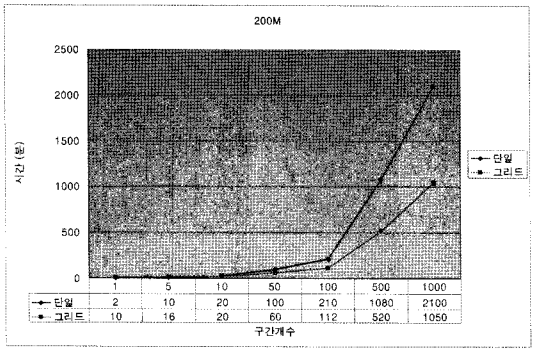

그림 6 200M 파일에서 실행 결과 (10Mbps)

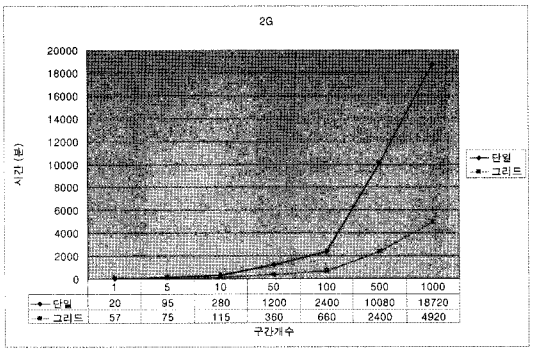

그림 7 2G 파일에서 실행 결과 (10Mbps)

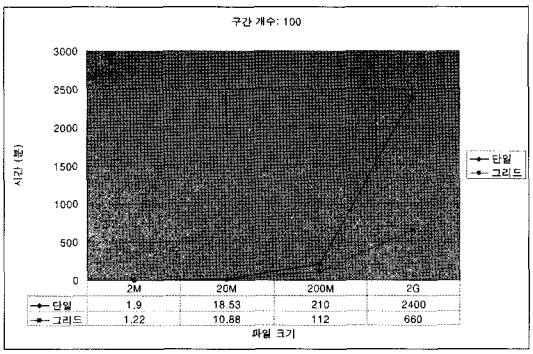

그림 8 100개 구간 관리 실행 결과 (10Mbps)

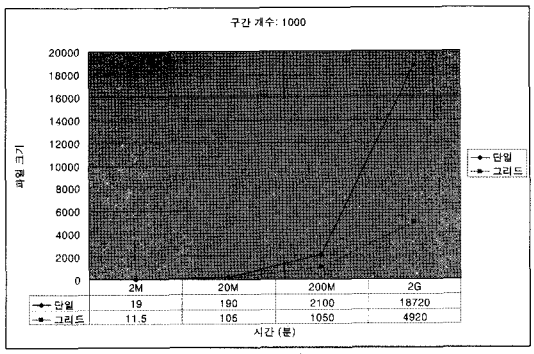

그림 9 1,000개 구간 관리 실행 결과 (10Mbps)

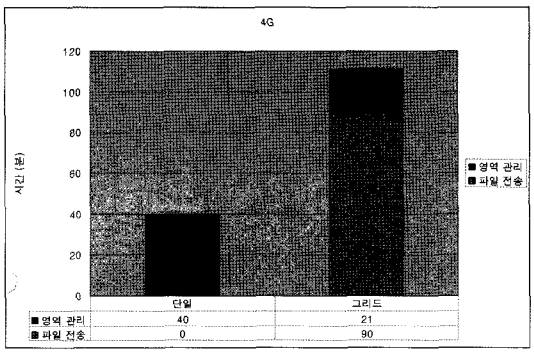

그림 10 4G 파일에서의 작업 별 소요 시간 (10Mbps)

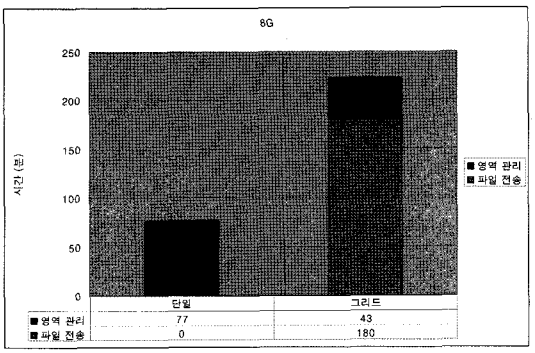

그림 11 8G 파일에서의 작업 별 소요 시간 (10Mbps)

나, 증가 비율은 단일 시스템이 그리드 시스템보다 더 높음을 확인할 수 있다. 이는 분산된 파일에서의 파일 영역 관리의 병행 실행이, 파일 크기의 증가로 발생되는 네트워크 전송 시간 (T_{upload} + $T_{download}$)의 증가를 충분히 상쇄할 만큼의 효과를 가져 오기 때문이다.

4G와 8G의 파일에 대해서도 동일한 실험을 진행하여, 유사한 결과를 뽑아 낼 수 있었다 (이들의 경우에는 실행에 시간이 많이 걸려서 구간 개수를 10개로 제한하여 실험하였다).

그림 10, 11에서는 구간 개수가 1개일 경우 파일을 그리드 시스템에 분산 전송하는데 소요되는 시간 (T_{upload}+$T_{download}$)이 전체 실행 시간 (T_{total}) 가운데 얼마나 차지하는지를 4G 및 8G 크기의 파일 영역 관리 테스트를 통하여 보여 준다. 두 경우 모두 전체 실행 소요 시간을 100으로 했을 때, 거의 80에 해당하는 부분이 네트워크 전송에 소요되었음을 알 수 있다. 따라서, 본 실험에서 사용한 10Mbps의 네트워크 환경 보다 더 빠른 전송 속도를 지원하는 네트워크 환경 (100Mbps 또는 1Gbps)에서 구축된 그리드 시스템에서는 전체 실행 시간이 네트워크 성능에 비례하여 단축 될 것이다.

5. 결 론

본 논문에서는 텍스트 파일 영역 관리를 그리드 서비스로 모델링 한 연구 결과와, 구현된 프로토타입의 실행 및 분석 결과를 기술하였다.

적은 수의 컴퓨터들로 구성된 그리드 환경에서 실행된 실험 결과 이지만, 본 연구를 통하여 영역 관리를 적용하는 파일이 대용량이고 변경되는 구간의 개수가 많다면 단일 시스템 보다 그리드 시스템에서 텍스트 파일 영역 관리를 하는 것이 효율적임을 알 수 있다.

앞으로, 본 연구에서 제시한 방법을 사용하여 다양한 통합 자동화 문제들에 적용하여 시스템 성능 개선을 평가 및 분석할 예정이다. 또한, 일시적인 네트워크 에러 등의 시스템 예외 사항들이 발생 하더라도 파일 영역 관리 서비스를 그리드 시스템에서 안정적으로 실행 할 수 있는 구조 설계에 대한 연구도 병행하여 진행할 계획이다.

참 고 문 헌

[1] Michael Sobolewski, Ravi-Kiran Malladi-Venkata, Abhijit Rai, Grid and Service-oriented Computing:

The Intergrid Perspective, *http://www.sorcer.cs.ttu.edu/publications/presentations/SORCER-GUG040616.ppt.*
[2] R.M. Kolonay and M. Sobolewski, Grid Interactive Service-Oriented Programming, *http://www.sorcer.cs.ttu.edu/publications/papers/CE2004-557.pdf.*
[3] 홍은지, 이세정, 이재호, 김승민, 다분야통합최적설계를 위한 데이타 서버 중심의 컴퓨팅 기반구조, *한국 CAD/CAM 학회 논문집*, 8권 4호: 231-242, 2003.
[4] iSIGHT, *http://engineous.com/product_iSIGHT.htm*
[5] ModelCenter, *http://www.phoenix-int.com/products/ModelCenter.php.*
[6] FiPER, *http://engineous.com/product_FIPER.htm*
[7] GridFTP: Protocol Extensions to FTP for the Grid
[8] Vijayshankar Raman, et al, Services for Data Access and Data Processing on Grids, DAIS Working Group, Global Grid Forum, February 9, 2003.
[9] Pradeep Padala (ed.), A Survey of Grid File Systems, Grid File System Working Group, Global Grid Forum.
[10] John H.Howard, An overview of the Andrew File System, *In Proceedings of the USENIX Winter Conference*, pages 23-2, Berkeley, CA, USA, January 1988, USENIX Association.
[11] B.Callaghan, B. Pawlowski, and P.Staubach, RFC 1813: NFS version 3 protocol specification, June 1995.
[12] S. Tuecke, et al, Open Grid Services Infrastructure (OGSI), *http://www.ggf.org/documents/final.htm.*
[13] I.Foster and C.Kesselman, Globus: A metacomputing infrastructure toolkit, *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115-128, 1997.

김 승 민

1995년 경북대학교 전자계산학과 학사 1997년 경북대학교 컴퓨터과학과 석사 1999년 3월~2001년 11월 삼성SDS㈜ 정보기술연구소 연구원. 2001년 12월~2005년 2월 한양대학교 최적설계신기술 연구센터 연구원. 1997년 3월~현재 서울대학교 컴퓨터공학부 박사과정. 관심분야는 XML, 그리드, WEB, 분산시스템, 최적설계

유 석 인

1977년 서울대학교 전기공학과 학사. 1980년 미국 Lehigh University 전산공학 석사. 1985년 미국 University of Michigan 전산공학 박사. 1985년~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 영상처리, 패턴인식, 기계학습, 그리드

김 일 곤

1980년 서울대학교 수학교육학과 학사 1988년 서울대학교 계산통계학과 전산학 전공 석사. 1991년 서울대학교 계산통계학과 전산학 전공 박사. 1997년 3월~1998년 8월 미국 죠지타운대학교 병원 ISIS 연구소 방문연구자. 1992년~현재 경북대학교 전자전기컴퓨터학부 교수. 관심분야는 지능형 에이전트시스템, 분산시스템, 의료정보학