그리드 컴퓨팅 환경에서의 상대성능지수에 기반한 작업 이주

(A Relative Performance Index-based Job Migration in Grid Computing Environment)

김 영 균 † 오 길 호 †† 조 금 원 ††† 고 순 흠 ††††
(Young-Gyun Kim) (Gil-Ho Oh) (Kum Won Cho) (Soon-Heum Ko)

요 약 본 논문에서는 글로버스(Globus) 기반의 MPICH-G2와 Cactus를 갖는 그리드 컴퓨팅 환경에서 작업 마이그레이션(Job Migration)에 대해 연구를 수행하였다. 그리드 컴퓨팅 환경에서 연산의 실행시간을 단축시킬 수 있는 연산자원이 풍부한 사이트를 찾아 작업을 마이그레이션 한다. 마이그레이션 사이트에서 체크 포인팅 파일에 기반 하여 연산의 수행이 중단된 지점부터 복구하여 연산을 재개한다. 마이그레이션 사이트를 선택하기 위해 사이트의 정적인 성능 지수와 CPU의 부하, 마이그레이션 작업을 전송하기 위한 네트워크의 부하를 고려한 작업 전송시간, 마이그레이션 사이트에서의 실행시간 예측 값을 사용하여 마이그레이션 이득이 큰 사이트로 작업을 마이그레이션 한다. 작업의 마이그레이션 시간과 실행시간 예측 값이 최소로 하는 사이트를 선택함으로서 보다 효율적인 그리드 컴퓨팅을 수행할 수 있도록 한다. 제안한 방법은 K*Grid 환경에서 전체 연산 시간을 효과적으로 단축함을 입증하였다.

키워드 : 작업 마이그레이션, 동적 그리드 컴퓨팅, 체크포인팅, 상대적인 성능지수

Abstract In this paper, we research on job migration in a grid computing environment with cactus and MPICH-G2 based on Globus. Our concepts are to perform job migration by finding the site with plenty of computational resources that would decrease execution time in a grid computing environment. The Migration Manager recovers the job from the checkpointing files and restarts the job on the migrated site. To select a migrating site, the proposed method considers system's performance index, cpu's load, network traffic to send migration job files and the execution time predicted on a migration site. Then it selects a site with maximal performance gains. By selecting a site with minimum migration time and minimum execution time, this approach implements a more efficient grid computing environment. The proposed method is proved by effectively decreasing total execution time at the K*Grid.

Key words : Job migration, Dynamic Grid computing, Checkpointing, Relative Performance Index

1. 서 론

최근 초고속의 통신망과 고성능 컴퓨팅 자원들의 일반화로 단일 사이트의 연산자원을 넘어서는 다중 사이트의 연산자원들을 효율적으로 이용하는 많은 그리드 컴퓨팅 시스템(Grid Computing System)들이 구축, 운영되고 있다. 이러한 대표적인 그리드 컴퓨팅 시스템으로서 국외의 경우 NEES(Network for Earthquake Engineering Simulation) Grid[1], Griphyn(Grid Physics Network)[2]가 있으며, 국내의 대표적인 K*Grid[3]가 있다.

그리드 컴퓨팅(Grid Computing)은 지리적으로 분산된 다중의 사이트에 구축된 HPC와 클러스터 컴퓨터가 초고속 인터넷으로 연결되어 대용량의 연산자원들을 요구하는 많은 응용들에 대해 연산자원들을 제공하는 것을 목표로 한다[4]. 단일 사이트의 HPC, 클러스터 시스템이 제공하지 못하는 대용량, 고속의 연산자원들을 제공하여 지구규모의 기상예측, 블랙홀의 충돌, 초신성의

† 학생회원 : 금오공과대학교 전자공학과
ygkim2004@paran.com
†† 종신회원 : 금오공과대학교 컴퓨터공학부 교수
gilho@knut.kumoh.ac.kr
††† 비 회 원 : 한국과학기술정보연구원 슈퍼컴퓨팅응용실 연구원
ckw@kisti.re.kr
†††† 비 회 원 : 서울대학교 기계항공공학부
floydfan@hanmail.net
논문접수 : 2005년 1월 18일
심사완료 : 2005년 5월 4일

폭발과 같은 천문학적 문제, 우주선, 항공기 등의 유체 역학 문제, 유전자 분석 등의 계산에 대해 널리 사용이 되고 있다[4,5].

본 논문에서는 그리드 컴퓨팅 환경에서 3차원 Euler 방정식을 이용하여 전산유체역학(CFD: Computational Fluid Dynamics) 문제를 계산하기 위해 Cactus[6] 프레임웍과 MPICH-G2를 사용하는 글로버스 기반의 그리드 컴퓨팅 환경에서 응용프로그램 수준의 체크포인팅 기법을 이용하여 작업 마이그레이션을 수행하는 동적 그리드 컴퓨팅(Dynamic Grid Computing)에 대해 연구하였다. Cactus 프레임웍은 슈퍼컴퓨터 또는 클러스터 컴퓨터 등의 환경에서 수행되는 과학자들과 공학자들을 위해 설계된 공개된 문제 해결 환경(Open Source Problem Solving Environments)을 제공한다[5].

그리드 컴퓨팅 환경에서 연산자원들의 효율적인 이용에 대한 많은 연구들이 수행되었다. 연산자원들의 효율적인 이용의 한 방법으로 고전적인 작업 마이그레이션 기법들이 그리드 컴퓨팅 환경에서 적용된 연구가 진행되었다[7-11]. 본 연구에서는 K*Grid 환경에서 3차원 Euler 방정식을 이용한 CFD 문제를 계산하는 Cactus 응용 프로그램의 연산시간을 단축하기 위하여 보다 연산자원이 풍부한 사이트로의 작업 마이그레이션을 수행한다. Cactus 응용 프로그램의 동적인 작업 마이그레이션을 구현하기 위해 응응 프로그램 수준의 체크포인팅 기법을 사용하여 주기적으로 연산 결과를 저장하였으며, 다중의 후보 사이트 중 이득이 가장 큰 마이그레이션 사이트를 선정하기 위하여, 사이트의 정적인 성능 지수인 프로세서의 처리 속도, 가용 가능한 프로세서의 개수, 유휴 주기억 메모리의 크기, 보조기억장치의 유휴 공간의 크기를 고려하였으며, 동적인 성능 지수로서 마이그레이션 사이트로의 프로세서 부하지수, 현재 사이트로부터 마이그레이션 사이트로의 통신 부하를 고려한 작업 전송시간, 마이그레이션 사이트에서의 실행 시간 예측을 고려하여 가장 이득이 큰 사이트로의 작업 마이그레이션을 수행하였다. 제안한 방법은 CPU 부하의 증가, 결함허용(Fault-tolerance)등의 연산시간의 증가요인으로부터 K*Grid 환경에서 작업의 연산시간을 효과적으로 단축함을 입증하였다.

2. 관련 연구

2.1 그리드 컴퓨팅과 작업 마이그레이션

그리드(Grid)는 몇몇 단체나 가상의 조직(Virtual Organization)의 요구를 서비스하기 위해 유지되는 컴퓨터와 기억장치 자원들의 분산된 집합이다[12,13]. 그리드 컴퓨팅은 문제들을 해결하기 위해 네트워크로 연결된 연산자원(Computational resources)들을 활용하는 것이다. 다중의 사이트에 분포되어 있는 많은 고성능의 연산 자원이 초고속의 통신망으로 연결된 그리드 환경에서는 다양한 사용자들의 요구에 따라 동적으로 변화하는 많은 연산자원들(예를 들어, 프로세서, 보조기억장치, 기타 그리드에 연결된 장치들)이 존재하며, 이러한 자원들을 효율적으로 사용하는 것은 중요하다. 그러한 방법들 중 하나로서 하나의 컴퓨터로부터 다른 컴퓨터로 작업들을 마이그레이션 하는 것은 매우 유용하다. 예를 들어서, 작업의 마이그레이션은 이동 에이전트(Mobile agents), 부하균등화(Load balancing), 주기적으로 마이그레이션 작업들을 정적인 기억장치로 기록함으로써 결함 허용(Fault-tolerance)을 구현하기 위해 사용되어 질 수 있다[14]. 작업이 시작될 때 가장 적합한 자원이 사용 중이고 몇 시간 동안 사용할 수 없다면 대안으로서 최적은 아니지만 좀 더 나은 자원이 사용될 수 있을 때까지 마이그레이션할 수 있다. 컴파일 시간에 알려지지 않은 필요한 자원들의 양은 동적으로 변화할 수 있다. 더 많은 메모리나 아니면 원격지에 떨어진 컴퓨터의 대규모 데이타 집합의 접근 등은 작업의 실행 시간에 발견 되어질 수 있다[15].

실행중인 작업을 마이그레이션하기 위해 작업의 현재 상태를 정적인 기억장치(하드디스크)로 기록하는 체크포인팅 기법을 흔히 사용한다. 체크 포인팅기법은 크게 시스템 수준의 체크포인팅(System-level checkpointing)과 사용자 정의 체크포인팅(User-defined checkpointing)으로 나누어진다. 시스템 수준의 체크 포인팅은 운영체제나 미들웨어에 의해 응용프로그램에 투명하게 자동적으로 제공되는 기법이다. 이것은 응용프로그램의 완전한 프로세스 이미지(Process image)를 캡처하는 것이다. 사용자 정의 체크포인팅은 응용프로그램의 상태에 대한 캡처를 프로그래머의 지원에 의존하는 기법이다.

시스템 수준의 체크포인팅은 이질적인 구조를 갖는 시스템 사이에 이식이 거의 불가능하다. 그러나 사용자 정의 체크 포인팅은 사용자가 시스템에 독립적인 파일 형식으로 저장할 수 있기 때문에 이식이 아주 용이하다. 시스템 수준의 체크 포인팅은 응용프로그램의 상세한 내용들을 모두 저장하기 때문에 파일의 크기가 대용량이다. 사용자 정의 체크포인팅은 재실행을 위해 요구되는 최소한의 체크포인트만을 저장하기 위해 사용자의 지원에 의존한다. 시스템 수준의 체크포인팅은 일정한 간격으로 체크포인팅을 수행하지만, 사용자 정의 체크포인팅은 프로그래머가 정의한 논리적인 상태에서 응용프로그램의 상태를 저장한다. 따라서 사용자에게 아주 높은 유연성을 제공한다. 이러한 이유로 사용자 수준의 체크포인팅은 후처리(Post-processing) 및 시각화(Visualization)와 같은 다른 용도들을 위해 사용될 수 있다[16].

동적인 그리드 컴퓨팅에서 Cactus의 연산 작업을 이주시키는 많은 연구들이 선행되었다. 대표적인 연구로서 Garbrielle Allen은 Cactus의 연산 작업을 응용프로그램 수준의 체크포인팅기법을 적용하여 글로버스 툴킷 기반의 그리드 환경에서 Cactus 응용프로그램의 마이그레이션을 시도하였다[8]. Sathish S. Vadhiyar는 GrADS 시스템에서 자원들의 부하 증가와 감소, 응용 프로그램이 실행되는 시점에서 시스템의 부하, 작업 마이그레이션으로 얻어지는 성능상의 이득을 고려하였다[9]. Carstten Ernemann은 계산 그리드(Computational Grid)상에서 단일 작업의 자원 소모를 다중 사이트 스케쥴링에 사용하는 적응형 다중 사이트 스케쥴링(Adaptive Multi-site Scheduling)기법을 제안하였다[10]. Chuang Liu는 Cactus 응용 프로그램에 대해 CPU의 부하, 메모리 크기, 분할된 작업들의 통신을 고려한 자원 할당에 대해 연구하였다[11].

2.2 글로버스와 MPICH-G2

그리드 환경의 이질성과 동적인 특성은 응용 프로그램의 개발을 어렵게 한다[7]. 글로버스는 분산된 자원들 간의 통신, 자원의 위치, 자원의 스케쥴링, 인증과 데이타 접근 등에 있어서 기본적인 능력과 인터페이스를 제공하는 메타컴퓨팅 기반 툴킷(Metacomputing infrastructure toolkits)이다[17]. MPICH-G2는 MPI(Message Passing Interface)의 그리드 컴퓨팅이 가능하도록 한 구현으로서 사용자가 병렬 컴퓨터상에 사용되는 동일한 명령어를 사용하여 동일하거나 다른 사이트에 위치한 다중 컴퓨터(Multiple Computers)들 간에 MPI 프로그램들을 실행할 수 있도록 한다. MPICH-G2는 글로버스 툴킷 서비스를 사용해서 인증, 허가, 프로세스의 생성, 모니터링, 제어, 통신, 표준 입출력 재지정, 원격 파일 접근 등의 목적을 위해 글로버스 툴킷 서비스를 사용함으로써 이질성을 숨긴다. 결과적으로 사용자는 다른 사이트에 있는 다중의 컴퓨터들을 통해 병렬 컴퓨터상에서 사용되는 동일한 명령어들을 사용해서 MPI 프로그램들을 실행할 수 있다[4]. 사용자는 컴퓨터들을 선택하기 위해 MDS(Monitoring and Discovery Service)[18]를 사용한다. 인증을 거친 후, 사용자는 MPI 연산 작업을 실행하기 위해 mpirun 명령을 사용한다. 이 명령은 작업을 기술하기 위해 RSL(Resource Specification Language)[19]를 사용한다. RSL 스크립트에 기초해서, MPICH-G2는 글로버스 툴킷이 갖는 DUROC(Dynamically-Updated Request Online Coallocator)[20] 라이브러리를 호출하여 사용자가 명시한 다양한 컴퓨터들 상에서 응용 프로그램을 스케쥴하고 시작시킨다. DUROC 라이브러리는 GRAM(Grid Resource Allocation and Management)[19] API와 프로토콜을 사용하여 각 컴퓨터당 하나씩의 부작업(subcomputation)을 시작하고 계속해서 관리한다[4]. GRAM은 GASS(Global Access to Secondary Storage)[21]를 사용하여 URL이 가리키는 원격지로부터 실행할 수도 있도록 한다. 또한 GASS는 응용프로그램이 시작된 후 표준 출력과 표준 에러(stdout과 stderr)를 사용자의 터미널로 전송한다.

2.3 Cactus의 특성 및 구조

특정 문제 해석을 위해 필요한 모든 계산적 편의를 제공하는 컴퓨터 시스템을 문제 풀이 환경(PSE : Problem Solving Environment)이라 지칭하며, Nimrod/G[22], Triana[23] 및 Cactus[5,6] 등이 현재 개발되어 있다. 이 중 Cactus는 기본적으로 천체물리학 연구자들의 공동 연구를 위한 기반으로서 개발되었으나, 천체물리학 연구 분야뿐만 아니라 전산유체역학 분야에서도 활용 가능하다.

Cactus는 기본적으로 각종 전산 분야의 기술들이 집약된 Flesh를 기반으로 하여 각 사용자의 단일 응용프로그램 해석자(Application solver), 체크포인팅을 위한 도구, 병렬 입출력을 위한 도구, 가시화 도구 등을 사용자의 필요에 따라 첨가하게 된다. 또한 Cactus는 단일 시스템뿐만 아니라 클러스터나 슈퍼컴퓨터 등 다양한 플랫폼에 구축될 수 있으며 현재 응용 연구자가 사용하는 도구를 쉽게 연결해 사용할 수 있는 환경을 제공한다. 예를 들어 글로버스 툴킷, HDF5 I/O, PETSc 과학 라이브러리, 가시화 도구 등을 연결하여 활용할 수 있다. Cactus는 응용과학자들이 일반적으로 활용하는 프로그래밍 언어 Fortran 77/Fortran 90, C, C++ 등의 프로그래밍언어를 지원하므로 수치해석을 위해 새로운 프로그래밍 언어를 배울 필요가 없다는 장점과, 컴퓨터 구조와 운영시스템에 영향을 받지 않고 실행 가능하다는 점, Cactus 프레임웍이 자동 병렬화를 담당하므로 사용자가 해석자의 병렬화를 위한 노력을 기울일 필요가 없다는 점, 객체지향적인 문제 풀이 환경을 제공하므로 다분야 연구자들이 공동으로 수치해석 코드를 개발 및 적용할 수 있다는 특징을 가진다. 그리고 최적화된 시뮬레이션 데이타의 전송법, 계산 결과의 시각화 등과 같은 부수적 작업에 대하여 각 분야의 전문가들이 Cactus 프레임웍에 맞추어 모듈을 제공하므로 응용 연구자는 자신의 특화된 문제 해석 자체에만 전념할 수 있다. Cactus의 기본 구조는 기본 프레임 위에 Flesh, Thorn, Driver 및 각종 연동 소프트웨어 등이 연결된 형태로 나타난다. Flesh는 사용자들의 Thorn, Driver 및 연동 가능한 소프트웨어 등 모듈의 형태로 존재하는 각 부분간의 상호작용을 가능하게 해준다.

Thorn은 주로 특정 응용 분야에 활용될 수 있는 객체들을 지칭하며, CFD 분야의 경우 기존의 수치해석자

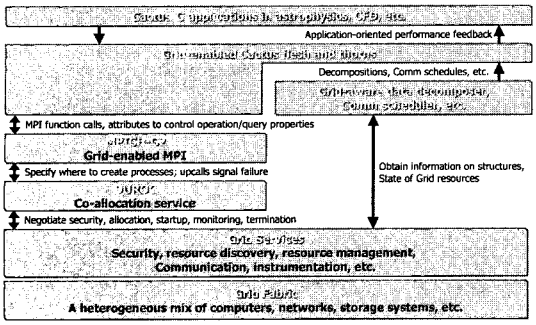

그림 1 MPICH-G2를 사용하는 그리드 환경에서의 Cactus 구조[7]

가 각 서브루틴별로 분할된 형태로 나타난다. Driver는 그리드 서비스 혹은 지역 클러스터(Local cluster)내에서 계산 격자를 분할하여 각각의 해석 자원에 할당하는 병렬처리를 지원한다. 연동 가능한 소프트웨어는 그리드 환경을 사용하기 위한 글로버스, 과학적 가시화를 위한 HDF5 혹은 연구자가 직접 개발한 소프트웨어를 사용할 수도 있다.

2.4 Cactus 기반의 CFD 해석과 3차원 Euler 방정식

2.4.1 지배 방정식 및 수치 기법

지배방정식으로서 3차원 압축성 Euler 방정식이 사용되었다. 3차원 Euler 방정식은 식 (1)과 같다.

$$\frac{\partial Q}{\partial t}+\frac{\partial E}{\partial x}+\frac{\partial F}{\partial y}+\frac{\partial G}{\partial z}=0 \quad \text{where}$$

$$Q=\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ \rho e\end{pmatrix},\ E=\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ \rho uw\\ (\rho e+p)u\end{pmatrix},\ F=\begin{pmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ \rho vw\\ (\rho e+p)v\end{pmatrix},$$

$$G=\begin{pmatrix}\rho w\\ \rho uw\\ \rho vw\\ \rho w^2+p\\ (\rho e+p)w\end{pmatrix} \qquad (1)$$

공간차분법으로는 AUSMPW+(modified Advection Upstream Splitting Method Pressure- based Weight function)[24]를 사용하였다. 시간 적분은 LU-SGS(Lower-Upper Symmetric Gauss Seidel)[25] 기법이 사용되었다. 시간 간격은 각 셀에서 선형화 행렬의 고유 값이 CFL 안정조건을 만족시키는 국소 시간 간격(Local time stepping)으로 크기를 결정하였다. 해석에 사용된 격자계는 body-fitted 좌표계를 사용하고 단일 블록으로 구성된 2차원 RAE-2822 Airfoil 형상 및 3차원 Onera-M6 날개 형상을 활용하여 일반 좌표계를 사용하는 격자 시스템에서의 Cactus를 활용하여 연산을 수행하였다.

2.4.2 3차원 Euler 방정식 수치해석자의 Cactus로의 적용

3차원 Euler 방정식을 위한 알고리듬의 일반적인 구

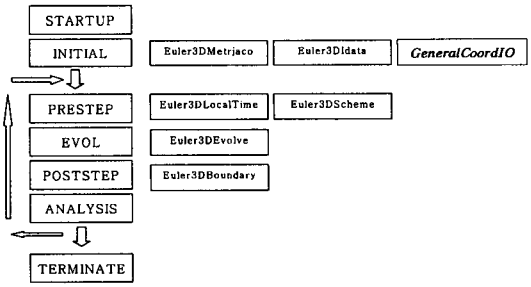

그림 2 3차원 Euler 방정식을 위한 타임 빈(Time Bin)과 Thorn

성은 전처리 단계에서 해석대상에 대한 격자생성(Mesh generation)한 후, 해석 단계에서는 유동조건의 초기화(Initial conditions) 및 격자 입력(Mesh input), 해석 격자를 계산 도메인(Computational domain)으로 전환(Metrics and jacobian)이 수행된 후 반복 계산이 실행된다.

반복 계산 영역은 경계조건 적용(Boundary conditions), 국소 시간전진 계산(Local time stepping), 공간차분에 의한 Flux 계산, 시간적분(Time integration) 등으로 구성된다. 이후 후처리 단계에서 결과 데이타에 대한 시각화 및 분석이 수행된다. 본 연구에서는 해석 단계에 포함되는 각각의 구성요소를 독립적인 Thorn으로 작성하여 Cactus가 지향하는 모듈화를 수행한다. 각각의 Thorn은 CCL 파일을 통해 상태정의(Configuration)를 하며, src/ 디렉토리 내에 수치해석코드의 구성 루틴을 저장한다. 기존의 수치해석 루틴은 Cactus에서 제공하는 파라미터(Parameter) 및 함수(Function)를 사용하기 위한 선언 추가, 기존 변수의 Cactus 변수(Cactus variables)로 변환 등의 설정 수행 후 저장된다. 그림 2는 3차원 Euler 방정식의 구성 루틴을 Thorn으로 제작하고, Cactus의 타임 빈에 맞추어 각각의 Thorn들을 스케쥴링한 흐름도이다. 초기 단계(CCTK_INITIAL)에서 body-fitted 격자 입력 및 metric과 jacobian 계산, 초기조건 생성을 수행하며, 초기 조건의 결과를 시간적분이 이루어지는 Euler3DEvolve에 초기 데이타로 저장하게 된다. 이후 iteration이 반복되는 CCTK_EVOL 타임 빈의 이전단계에서 국소 시간 간격과 플러스를 계산하고, Evolution step에서 새로운 유동물성치를 계산하며, EVOL 다음 단계에서 경계조건이 적용된 후, 이 과정을 반복한다. 프로그램의 종료를 위해서는 임의로 정하는 반복회수(Number of iteration)와 전역시간(Global time)에 의한 두 가지 방법이 있다. 그러나 정상상태의 해를 얻기 위한 국소 시간 간격을 적용한다면 반복회수에 의해 프로그램의 종료가 제어된다.

3. 제안한 작업 마이그레이션 기법

제안한 작업 마이그레이션 기법은 현재 사이트와 마이그레이션할 사이트의 현시점에서 CPU의 부하, CPU의 성능, 유휴 프로세서 수, 유휴 메모리, 네트워크 부하, 마이그레이션할 사이트에서 잔여 연산을 계산하기 위한 예측된 실행 시간을 기반으로 작업 마이그레이션을 실행할 것인지 판단한다. CFD 문제는 연산 작업이 연산식을 반복적으로 계산함으로서 결과를 얻는다. 따라서 실행시간을 추정하는 것이 비교적 용이하다. 즉 연산의 특정 반복 회수에 수행된 시간을 측정함으로써 나머지 연산의 반복 회수에 수행되는 실행시간을 추정할 수 있다. 그러나 마이그레이션 할 사이트에서는 연산의 특정 반복 회수에 수행된 시간을 측정한 값이 없으므로, 현재 사이트의 연산 능력과 마이그레이션할 사이트의 연산 능력의 상대적인 비를 구하여 현재 사이트에서 연산의 특정 반복 회수의 수행에 걸린 측정된 시간 값을 곱함으로써 마이그레이션할 사이트에서의 대략적인 연산 시간을 추정할 수 있다. 제안한 방법의 기본적인 개념을 정량화하기 위하여 본 논문에서는 실행시간을 추정하기 위해 식 (11)로부터 계산된 성능 지수 P_m의 값을 사용한다.

마이그레이션할 사이트와 현재 사이트와의 대략적인 상대적인 성능 지수 P_m은 식 (2)와 같이 계산한다. 현재 노드의 현시점에서의 평균 프로세서 부하를 L_c, CPU 속도를 $S_{c.cpu}$, 유휴 CPU 개수를 $N_{c.cpu}$라 하고, 마이그레이션할 사이트의 현 시점에서의 평균 프로세서 부하를 L_m, CPU 속도를 $S_{m.cpu}$, 유휴 CPU개수를 $N_{m.cpu}$ 이라 하면, 현재 사이트에서 사용할 수 있는 연산 능력은 부하를 제외한 값이므로 $(1-L_c)$ 이고, 마이그레이션 할 사이트에서의 가용할 수 있는 연산 능력도 부하를 제외하면 $(1-L_m)$이 된다. $L_{c.free}=(1-L_c)$, $L_{m.free}=(1-L_m)$라 할 때, $L_{m.free}=0$라는 것은 마이그레이션 할 사이트에 연산 능력에 여유가 없다는 것을 뜻한다. $L_{c.free}=0$이라는 것은 현재 노드가 과부하 상태라는 의미를 갖는다. 따라서 사용 가능한 연산 능력의 상대적인 성능 지수 P_m은 식 (2)와 같이 계산된다. (단, $0 \le L_m, L_c \le 1$)

$$P_m = \frac{L_{m.free} \times S_{m.cpu} \times N_{m.cpu}}{L_{c.free} \times S_{c.cpu} \times N_{c.cpu}} \quad (2)$$

$$\text{if} \quad L_{c.free} = 0, then \quad P_m = L_{m.free} \times S_{m.cpu} \times N_{m.cpu}$$

현재 사이트에서 3차원 Euler 방정식을 이용한 CFD 문제의 추정 연산 실행 시간(Estimated Execution Time) $T_{e.c}$는 남아 있는 연산의 반복 회수를 C_{iter} 현재 사이트에서의 체크포인팅시간을 포함한 연산의 특정 반복회수 n에 소요된 시간을 $T_{n.c}$ 라 할 때

$(T_{n.c} = T_{checkpointing} + T_{computation.n})$

$$T_{e.c} = \frac{C_{iter}}{n} \times T_{n.c} \quad (3)$$

마이그레이션할 사이트에서의 추정 연산 실행시간 (Estimated Execution Time) $T_{e.m}$은

$$T_{e.m} = \frac{\frac{C_{iter}}{n} \times T_{n.c}}{P_m} = \frac{T_{e.c}}{P_m} \quad (4)$$

현재 사이트와 마이그레이션할 사이트와의 현재 네트워크 부하를 고려하여 W bytes를 전송하는 데 소요되는 시간을 T_w, 마이그레이션할 작업 화일의 크기를 $S_{job.m}$ bytes라 할 때,

현재 사이트에서 마이그레이션할 사이트로의 작업 마이그레이션 소요 시간 T_{jm}은

$$T_{jm} = T_w \times \frac{S_{job.m}}{W} \quad (5)$$

마이그레이션을 했을 때 총 연산 소요 시간 $T_{total.migration}$은

$$T_{total.mogration} = T_{jm} + T_{e.m} \quad (6)$$

따라서, 현재 사이트에서 마이그레이션할 사이트로의 작업 마이그레이션은 식 (7)을 만족할 때 실행한다.

$$T_{e.c} > T_{total.mogration} \quad (7)$$

현재 사이트의 프로세서의 가용능력과 마이그레이션할 사이트의 프로세서의 가용 능력을 상대적인 비율로 간략히 표현한 식 (2)는 유휴 주기억 용량과 보조기억장치 용량을 고려할 때, 식 (8)과 같이 확장 될 수 있다. 현재 노드의 유휴 주기억 용량을 $M_{c.freememory}$, 보조기억장치의 용량을 $S_{c.freedisk}$, 마이그레이션할 노드의 유휴 주기억 용량을 $M_{m.freememory}$, 보조기억장치의 용량을 $S_{m.freedisk}$, α, β, γ 는 가중치로서 $0 \le \alpha, \beta, \gamma \le 1$이라 할 때, P_m의 값에 프로세서의 성능, 유휴메모리의 양, 유휴 하드디스크의 양을 어느 정도로 반영할 것인가를 제한하는 가중치이다. 일반적으로 연산속도에 미치는 영향이 프로세서의 성능>주기억 용량>보조기억장치의 용량으로 가정할 때, $\alpha > \beta > \gamma$ 의 조건을 만족하는 값을 선택한다.

$$P_m = \alpha \frac{L_{m.free} \times S_{m.cpu} \times N_{m.cpu}}{L_{c.free} \times S_{c.cpu} \times N_{c.cpu}} + \beta \frac{M_{m.freememory}}{M_{c.freememory}} + \gamma \frac{S_{m.freedisk}}{S_{c.freedisk}} \quad (8)$$

최초의 연산 작업을 할당하기 위한 사이트를 선택하고자 할 때, 특정 회수의 연산 소요 시간을 측정한 값이 없으므로, 현재 사이트와 마이그레이션할 사이트의 상대적인 성능지수인 P_m과 네트워크 부하를 고려한 작업 전송시간 T_{jm}를 사용하여 작업을 할당한다. 따라서 식 (9)을 만족하는 사이트를 최초의 작업을 할당하기 위한 사이트로 선택한다.

```
Step 1:Repeat
Step 2: L ← { all of available sites in a grid computing }
Step 3: Tmax = 0
Step 4: For each m ∈ L Do
        Begin
Step 5:  Get current site and migrating site m's CPU load value,
          free memory size(main memory and disk).
Step 6:  Compute Pm from Step 5's values.
Step 7:  Checkpointing current job's state to files,
          get remaining iteration numbers of current computation and Tm,c
           and compute Te,c and Te,m.
Step 8:  Get network's load value between current site and migrating site
           and compute Tjm and Ttotal,computation.
Step 9:  If Tmax < ( Te,c - Ttotal,migration ) ) Then
Step 10:        Tmax = Te,c - Ttotal,migration
Step 11:        Migration_Site ← m
          End if
        End for
Step 12: If Tmax ≠ 0 Then
Step 13:  Select Migration_Site m with Tmax and
           compute new Tjm from current Tjm.
Step 14:  The current site's process terminates and migrating from current site
           to the selected site m and resuming the computation on the selected
            site m.
        End if
Step 15: Until CFD computation terminates.
```

알고리즘 1 제안한 방법의 CFD 연산 작업의 마이그레이션 알고리즘

$$Min(\frac{1}{P_m} + T_{jm}) \tag{9}$$

작업 할당 후 식 (7)을 만족하는 연산 자원이 보다 풍부한 다른 사이트가 존재할 경우, 동적 마이그레이션을 수행한다. 식 (7)의 조건을 만족하는 사이트가 다수개 존재하는 경우, 전체 연산 시간을 가장 많이 단축할 수 있는 마이그레이션의 이득이 가장 큰 식 (10)를 만족하는 사이트로 마이그레이션을 수행한다.

$$Max(T_{e,c} - T_{total,migration}) \tag{10}$$

클러스터 시스템 내의 프로세서들 사이를 연결한 네트워크의 통신 속도가 빠를수록 마이그레이션 했을 때 이득이 크다. 또한 연산에 참여 가능한 가용 프로세서의 비트수가 클수록 마이그레이션의 이득이 크다. 현재 사이트의 클러스터 시스템의 가용 프로세서를 연결하는 네트워크의 전송 속도를 $S_{c,connection}$, 마이그레이션할 사이트의 클러스터 시스템의 가용 프로세서의 전송 속도를 $S_{m,connection}$이라 하고, 마이그레이션할 사이트의 가용 프로세서의 비트수를 $B_{m,cpu}$, 현재 사이트의 가용 프로세서의 비트수를 $B_{c,cpu}$, 현재 사이트의 성능지수와 마이그레이션할 사이트의 성능 지수 비는 식 (2)와 식 (8)로부터 식 (11)로 확장할 수 있다.

$$P_m = \alpha(\frac{L_{m.free} \times S_{m.cpu} \times N_{m.cpu}}{L_{c.free} \times S_{c.cpu} \times N_{c.cpu}} \times \frac{S_{m.connection}}{S_{c.connection}} \times \frac{B_{m.cpu}}{B_{c.cpu}}) + \beta\frac{M_{m.freememory}}{M_{c.freememory}} + \gamma\frac{S_{m.freedisk}}{S_{c.freedisk}} \tag{11}$$

P_m의 값에 따라서, $P_m < 1$를 만족하는 경우, 현재 사이트보다 가용 연산 능력이 떨어지는 사이트로의 이동이다. $P_m = 1$를 만족하는 경우, 동일한 연산 능력을 갖는 노드로의 이동이며, $P_m > 1$를 만족하는 경우, 현재 사이트보다 연산 능력이 우수한 사이트로의 이동이다. 따라서 결함 허용을 제외하면 일반적으로 $P_m > 1$인 사이트로의 마이그레이션을 가정한다. 또한 연산 자원들이 동적으로 변화하는 그리드 컴퓨팅 환경에서는 연산 자원이 보다 풍부한 사이트가 연속적으로 존재할 수 있게 된다. 이 경우, 연속적으로 마이그레이션이 발생하는 마이그레이션의 도미노 효과(Domino effect)가 발생할 수 있다. 또한 두개의 사이트 사이(또는 두개의 클러스터 시스템 사이)를 번갈아 가면서 연속적으로 마이그레이션 하는 핑퐁효과(Pingpong effect)도 발생할 수 있다. 이러한 현상은 빈번하게 마이그레이션 함으로써 작업 전송 시간의 증가로 인하여 전체 연산 시간을 증가시키게 된다. 따라서, 이러한 부작용을 방지하기 위해 식 (6)의 T_{jm}은 이전 단계까지 마이그레이션의 작업전송 시간을 누적하여 사용하도록 식 (12)와 같이 수정할 수 있다.

n번째 마이그레이션 할 사이트에서의 T_{jm}은

$$T_{jm} = \sum_{i=1}^{n} T_{i,jm} \quad (12)$$

식 (12)의 T_{jm}을 사용하여 $T_{total,migration}$의 값을 구함으로써 n-1번째까지 작업을 마이그레이션하기 위해 소요된 시간을 전체 연산시간에 누적하게 된다. 따라서, 작업의 전체 연산시간에 마이그레이션으로 소요된 총 작업 전송 시간을 포함하게 됨으로써 과도한 마이그레이션이 발생하는 것을 방지한다. 또 다른 방법으로는 마이그레이션 회수를 제한하는 방법이 있다.

제안한 기법을 사용한 CFD 연산 작업의 마이그레이션 알고리즘은 알고리즘 1과 같다. 알고리즘 1은 CFD 연산 작업을 종료할 때까지 그리드 컴퓨팅에 참여하는 유효한 사이트의 개수인 | L | 의 값만큼 반복 수행하며 P_m 값을 평가한다.

4. 구현

제안한 방법은 그림 3에서 Migration Server와 Migration Manager로 Perl v5.6.1을 사용하여 레드햇 리눅스(Redhat Linux)7.3 운영체제에서 구현하였다. 실험을 하기 위해 사용한 3차원 Euler 방정식은 Fortran 77로서 Cactus 4.0 beta 12 프레임웍에서 구현 되었다. Migration Server와 K*Grid gateway는 작업전송의 보안을 위해 SSH의 RSA키를 생성하여 인증과정을 거친다. 또한 K*Grid에 참여하는 모든 연산 자원들도 GSISSH의 RSA키를 사용하여 인증한다.

그림 3에서와 같이 사용자는 Migration Server를 통해 K*Grid환경에서 실행될 사용자의 Cactus 응용 프로그램을 제출, 관리하고 K*Grid Gateway를 통해서 MPICH-G2 환경에서 실행될 Cactus 응용 프로그램을 Migration Manager를 통해 연산자원이 있는 적절한 사이트를 선정하여 클러스터 시스템으로 전송한다. 또한 마이그레이션 사이트에서 실행중인 Cactus 작업에 대해 웹을 통해 실시간 모니터링을 수행한다. 그리드 컴퓨팅에 참여하는 사이트에서 실행중인 Migration Manager는 Migration Server 또는 다른 사이트의 Migration Manager로부터 전송된 작업을 실행, 중단, 새로운 마이그레이션 사이트 선정(Resource selector), 마이그레이션 여부를 판단(Migration Logic)하여 다른 사이트로 작업을 마이그레이션 하거나 작업에 대한 실시간 모니터링 정보를 Migration Server에게 제공하는 역할(Monitor)을 수행한다. Migration Manager는 주기적으로 현재 사이트의 부하 정보와 현재 사이트에서 수행중인 Cactus 작업의 잔여 연산회수를 수집하여 제안한 방법의 연산식 10을 만족하는 사이트를 선택한다. K*Grid 내의 연산자원들에 의해 수행된 최종 연산 결과는 연산의 마지막 단계에서 작업을 제출한 Migration Server로 전송된다.

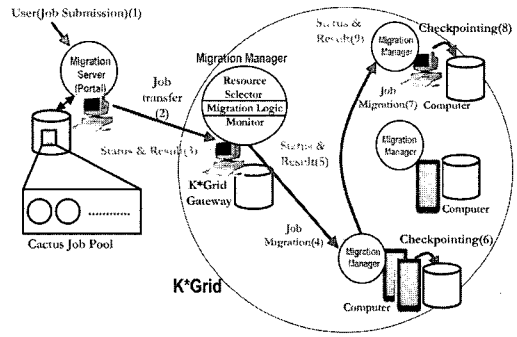

그림 3 Cactus Job의 체크포인팅과 마이그레이션 과정

그림 3에서 K*Grid Gateway는 K*Grid에 연결된 연산자원들을 사용하기 위한 인증 서버 역할을 수행한다. K*Grid내의 연산자원들은 지역적으로 분산되어 있으며 초고속의 통신망으로 연결되어 있다. Migration Server 역할을 수행하는 컴퓨터는 Venus 클러스터 시스템과 함께 KISTI내에 위치한다.

K*Grid중 실험에 사용한 클러스터 시스템 Venus, Jupiter, Newcluster는 그림 4와 같이 위치한다. Venus와 Jupiter는 100Mbps Fast Ethernet으로 연결된 각각 0.1Km,0.15Km내의 LAN으로 연결되어 있으며, 이들과 Newcluster는 140Km의 거리를 1Gbps의 Gigabit Ethernet으로 연결되어 있는 WAN 환경을 갖는다. 컴퓨터의 정적인 성능 지수는 계산의 편리를 위해서 그리드 컴퓨팅에 참여할 사이트내의 프로세서의 성능 지수를 기준값으로 나눈 값을 사용한다.

성능 지수의 기준 값으로서 CPU Clock 속도 $S_{basis,cpu}$=1GHz, 주기억장치의 용량 $M_{basis,freememory}$=512Mbytes, 하드디스크 용량 $S_{basis,freedisk}$=40Gbytes, 클러스터 내부의 가용프로세서를 연결하는 네트워크의 전송 속도$S_{basis,connection}$=1Gbps, 외부 사이트와 연결하는 네트워크의 전송속도 $S_{basis,network}$=100Mbps, 가용 프로세서의 비트 수 $B_{basis,cpu}$=32Bits를 사용한다.

표 1의 정적인 성능지수 중 $M_{c,freememory}$, $S_{c,freedisk}$는

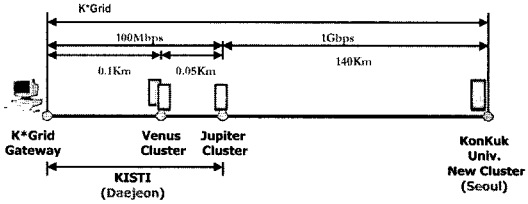

그림 4 작업 마이그레이션 실험에 사용한 K*Grid

표 1 K*Grid의 KISTI Venus, Jupiter, 건국대 클러스터 시스템의 기준값에 대한 성능지수

성능지수	KISTI Venus	KISTI Jupiter	KonKuk Univ. Newcluster
$S_{c,cpu}$	2.0	1.7	4.0(2.0×2)
$N_{c,cpu}$	0~63	0~16	0~7
$M_{c,freememory}$	0~1.0	0~2.0	0~2.0
$S_{c,freedisk}$	0~1.0	0~2.0	0~0.4
$S_{c,connection}$	0.1	0.1	1.0
$B_{c,cpu}$	1.0	1.0	1.0
$S_{c,network}$	0.1	0.1	1.0

시스템에서 실행중인 프로세스(process)에 따라서 동적으로 변화하는 값이다. 해당 사이트의 유휴 프로세서의 개수, 유휴 메모리 용량, 유휴 하드디스크 용량과 프로세서의 평균부하를 구하여 프로세서의 상대적인 성능지수 P_m을 구한다. 각 사이트의 Migration Manager는 해당 사이트의 성능 지수 값들을 유지하고 이들 값을 요청하는 다른 Migration Manager에게 값을 제공한다. 표 2는 유휴 주기억 용량, 유휴 하드디스크 용량이 P_m 값에 미치는 영향이 극히 작은 경우, 부하와 프로세서의 성능지수만을 반영하기 위해 가중치 $\alpha = 1.0, \beta = 0, \gamma = 0$ 를 사용하고, Venus의 $N_{c,cpu}$=16일 때, Venus로부터

표 2 표 1의 성능지수 값에 따른 $N_{c,cpu}$=16일 때, Venus의 L_c에 대한 Newcluster의 P_m값

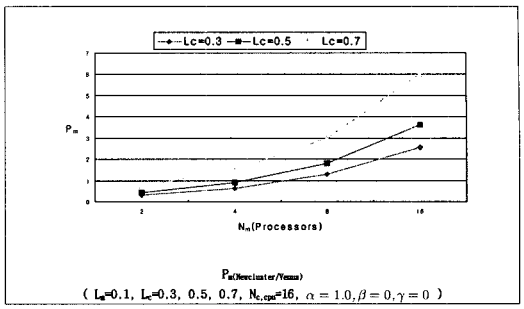

표 3 실험에 사용한 K*Grid의 KISTI Venus, Jupiter, 건국대 클러스터 시스템

Organization		KISTI	KISTI	KonKuk Univ.
Model		Venus	Jupiter	Newcluster
Architecture		Linux Cluster	Linux Cluster	Linux Cluster
OS		Redhat Linux 7.3	Redhat Linux 7.3	Redhat Linux 7.3
CPU	CPU	Intel Pentium®Ⅳ	Intel Pentium®Ⅳ	Intel Pentium®Ⅳ
	Clock	2.0 GHz	1.7GHz	2.0GHz
	#CPU/Node	1	1	2
	#Node	63	16	7
	Total	63	16	14
RAM	#RAM/Node	512MB	1GB	1GB
	Total	31.5GB	16GB	7GB
Hard Disk	#Hard Disk/Node	40GB	80GB	16GB
	Total	40GB+500GB(nfs)	80GB+500GB(nfs)	16GB
Network	Login node	venus	jupiter	newcluster
	Host name	ve001 ~ ve063	ju001 ~ ju 016	node2 ~ node8
	Domain name	gridcenter.or.kr	gridcenter.or.kr	konkuk.ac.kr
	Interface	Fast Ethernet (100Mbps)	Fast Ethernet (100Mbps)	Gigabit Ethernet (1Gbps)

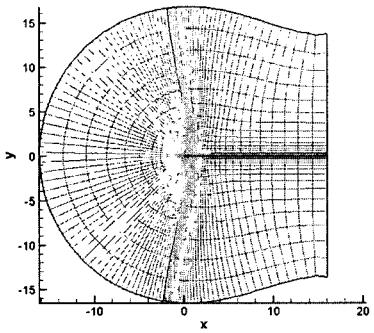

그림 5 RAE-2822 해석 격자계

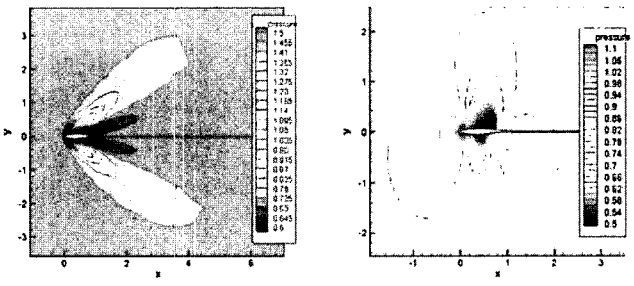

그림 6 초음속(L) 및 천음속(R) 유동 해석 결과

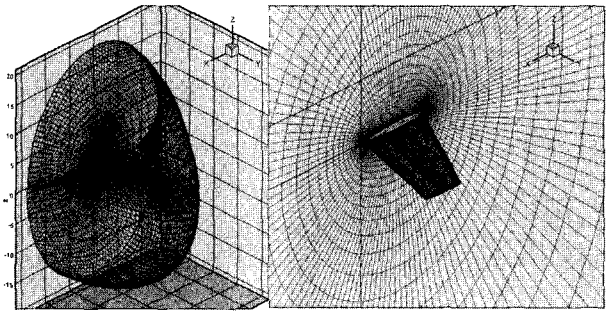

그림 7 3차원 Onera-M6 날개 해석 격자계(O-type)

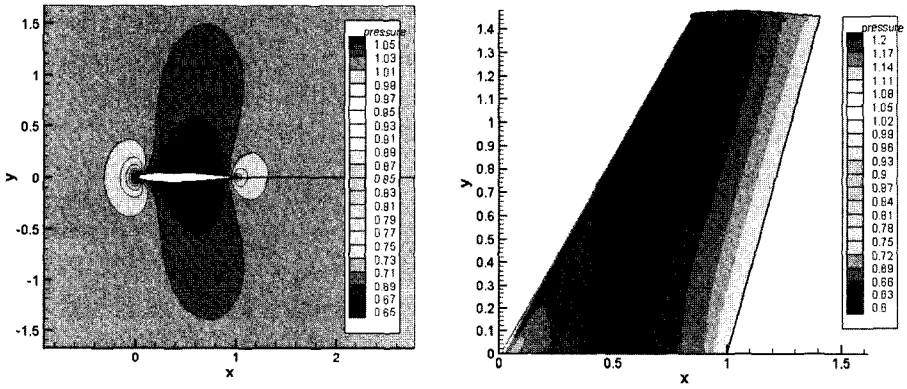

그림 8 압력 분포(날개 뿌리 단면 및 표면 압력)

표 4 마이그레이션에 사용한 작업 파일

작업	파일 설명	파일 구분	파일 크기	작업 총크기
Job 1 (161×41×3개의 격자)	실행코드 파일	cactus_m7	2,268Kbytes	5.113Mbytes
	격자좌표 파일	.grd	211.2Kbytes	
	파라미터 파일	.par	1.410Kbytes	
	RSL 파일	.rsl	0.420Kbytes	
	체크포인팅 파일	.asc×7	376Kbytes×7	
Job 2 (143×33×65개의 격자)	실행코드 파일	cactus_m9	2,265Kbytes	54.14Mbytes
	격자좌표 파일	.grd	11,066.668Kbytes	
	파라미터 파일	.par	1.411Kbytes	
	RSL 파일	.rsl	0.420Kbytes	
	체크포인팅 파일	.asc×7	5830Kbytes×7	
Job 3 (161×41×100개의 격자)	실행코드 파일	cactus_m11	1,699.693Kbytes	89.734Mbytes
	격자좌표 파일	grd	211.2Kbytes	
	파라미터 파일	.par	1.411Kbytes	
	RSL 파일	.rsl	0.420Kbytes	
	체크포인팅 파일	.asc×7	12,546.027Kbytes×7	

Newcluster로 마이그레이션을 수행하기 위한 P_m의 값을 나타낸 것이다. Venus의 L_c가 증가함에 따라 $P_m > 1$가 되는 Newcluster의 프로세서 개수가 감소함을 알 수 있다. 즉, Venus의 부하가 증가할 경우, 상대적으로 Newcluster의 유휴 프로세서의 개수가 줄어들어도 마이그레이션을 수행하기 위한 P_m의 값을 갖는 것을 알 수 있다.

5. 실험환경 및 결과

실험은 Globus 2.2.4와 MPICH-G2 1.2.5-1a를 사용하는 K*Grid에서 수행하였다. 실험에 사용한 K*Grid의 KISTI Venus, Jupiter, 건국대의 Newcluster 시스템의

사양은 표 3과 같다.

그림 5에 나타난 바와 같은 RAE-2822 형상의 airfoil을 해석하는 것으로 유동해석 결과는 그림 6과 같다. 본 형상은 기본적으로 2차원 유동을 나타내며, 161×41개의 격자점을 가진다. 그러나 현재의 Cactus 프레임웍이 3차원 해석 전용으로 개발되었으므로 z-방향으로 격자를 뽑아내어 총 161×41×3개의 격자점을 가지는 문제를 해석하도록 하였다. 3차원의 Onera-M6 비행기 날개 형상에 대한 병렬 해석을 수행하였다. 이 경우 총 격자점은 그림 7과 같이 143×33×65개를 갖으며 연산 결과 나타나는 압력 분포는 그림 8과 같다. 총 격자점은 143×33×65개이다. 작업 마이그레이션을 수행하기 위한 Cactus 작업의 크기는 Job 1의 경우 3차원 Euler 방정식을 풀기 위한 실행코드 파일, 3차원 Euler 방정식의 체크 포인팅 파일, 불균일한 격자 좌표를 담고 있는 격자좌표 파일인 .grd화일, 실행에 필요한 파라미터 파일인 .par파일, 연산자원에 대한 구성을 기술한 파일인 .rsl로 구성되어 있다. Job 1과 Job 2, Job 3의 필요한 작업 파일의 구성과 크기는 표 4와 같다.

그림 9는 K*Grid Gateway에 제출된 작업이 K*Grid Gateway로부터 Venus 클러스터, Newcluster 클러스터 시스템 순서로 각 사이트에서 1분의 시간 할당량(time quantum)을 가지고 제출된 작업을 실행하여 마이그레이션 하는 과정을 웹을 통해 실시간 모니터링한 결과를 보여준다. 표 5는 K*Grid Gateway을 통해 제출된 작업이 Venus→ Newcluster, Venus→ Jupiter로의 Job 1, Job 2, Job 3의 작업 마이그레이션 소요시간을 10회 측정한 평균값을 보여 주고 있다. K*Grid Gateway→Venus의 경우 100Mbps Fast Ethernet으로 연결되어 있고, Venus→Newcluster는 1Gbps의 Gigabit Ethernet으로 연결되어 있다. 물리적인 전송 속도는 K*Grid Gateway→Venus보다 Venus→Newcluster가 10배정도 빠르지만, 실험에 사용한 작업의 마이그레이션 속도는 Venus→Newcluster사이가 3.11배정도만 빠르다. 작업파일의 크기가 증가 할수록 작업 마이그레이션 시간도 증가하지만 실제 연산시간에 비해 작업 마이그레이션 소요시간은 큰 비중을 갖지 않음을 알 수 있다. 표 6과 7은 K*Grid Gateway에 제출된 작업이 Venus→Newcluster와 Venus→Jupiter로 마이그레이션한 경우의 실제 연산 소요 시간을 보여 주고 있다. 표 6은 Job 1의 Venus로부터 Jupiter, Newcluster로 마이그레이션 하는 경우의 수행시간을 보여준다. Newcluster로 마이

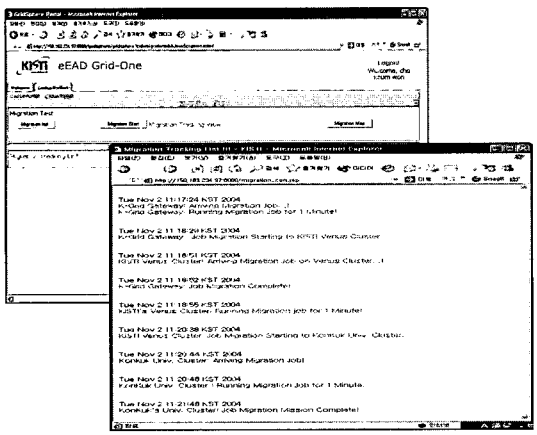

그림 9 K*Grid에 제출된 작업의 웹을 통한 실시간 모니터링

표 5 Job 1, Job 2, Job 3의 작업 마이그레이션 소요시간

Job	Migration Site	소요시간(10회 평균 값)
Job 1	K*Grid Gateway→Venus	6.9초
	Venus→Newcluster	2초
	Venus→Jupiter	5.1초
Job 2	K*Grid Gateway→Venus	26초
	Venus→Newcluster	11.2초
	Venus→Jupiter	30.3초
Job 3	K*Grid Gateway→Venus	41.2초
	Venus→Newcluster	20초
	Venus→Jupiter	49.1초

표 6 Venus로부터 Newcluster, Jupiter로 마이그레이션한 Job 1의 연산시간

Migration Site	L	N_{cpu}	P_m ($\alpha=0.7, \beta=0.2, \gamma=0.1$)	실제 연산 소요시간
Venus	L_c=0.52	$N_{c.cpu}$=4	3.90	627분14.0초
Newcluster	L_m=0.02	$N_{m.cpu}$=4		380분32.1초
Jupiter	L_m=0.12	$N_{m.cpu}$=4	0.874	1783분12초

표 7 Venus로부터 Newcluster, Jupiter로 마이그레이션한 Job 2의 연산시간

Migration Site	L	N_{cpu}	P_m ($\alpha=0.7, \beta=0.2, \gamma=0.1$)	실제 연산 소요시간
Venus	L_c=0.17	$N_{c.cpu}$=8	1.65	715분48.1초
Newcluster	L_m=0.01	$N_{m.cpu}$=8		603분38.4초
Jupiter	L_m=0.10	$N_{m.cpu}$=8	0.64	1150분9.9초

표 8 Venus로부터 Newcluster, Jupiter로 마이그레이션한 Job 3의 연산시간

Migration Site	L	N_{cpu}	P_m ($\alpha=0.7, \beta=0.2, \gamma=0.1$)	실제 연산 소요시간
Venus	L_c=0.014	$N_{c.cpu}$=4	1.98	2384분59초
Newcluster	L_m=0.025	$N_{m.cpu}$=4		2279분48초
Jupiter	L_m=0.0001	$N_{m.cpu}$=4	0.87	2740분55초

그레이션 하는 경우 247분정도의 수행시간 단축을 할 수 있음을 알 수 있다. 표 7은 Job 2의 Venus로부터 Jupiter, Newcluster로 마이그레이션 하는 경우의 수행시간을 보여준다. Newcluster로 마이그레이션 하는 경우 112분의 수행시간 단축을 보여 주고 있다. 표 8은 Job 3의 Venus로부터 Jupiter, Newcluster로 마이그레이션하는 경우의 수행시간을 보여 준다. Newcluster로 마이그레이션 하는 경우 105분의 수행시간 단축을 보여 주고 있다.

체크포인팅 기법을 사용한 작업 마이그레이션은 동적으로 변화하는 그리드 컴퓨팅 환경에서 장시간 동안 수행되어야 하는 과학기술분야의 연산 작업이 CPU 부하 증가, 결함허용 등 작업의 연산시간을 증가시키는 요인으로부터 영향을 적게 받으며 전체 연산시간을 단축할 수 있음을 알 수 있다.

6. 결론 및 향후 연구

본 논문에서는 리눅스 클러스터(Linux Cluster)컴퓨터로 구성된 글로버스 기반의 그리드 컴퓨팅 환경에서 Cactus와 MPICH-G2를 사용하여 3차원 Euler 방정식을 이용한 CFD 문제를 해결하였다. 또한 마이그레이션 사이트를 선택하기 위해 시스템의 성능지수와 CPU 부하, 네트워크 부하를 고려한 작업 전송시간, 실행시간 예측에 기반 하여 마이그레이션 이득이 큰 사이트로 작업을 마이그레이션 하는 연구를 수행하였다. 응용프로그램 수준의 체크포인팅 기법을 이용하여 보다 풍부한 연산 자원이 있는 사이트로 동적으로 이동하여 중단된 지점부터 계속적인 연산을 수행함으로써 작업 시간을 단축할 수 있음을 보여 주었다. 특히 현재 사이트의 CPU 부하가 급격히 증가하여 연산시간이 길어지는 경우, 유휴 사이트로 이동하여 연산을 수행함으로써 연산 시간을 효과적으로 단축 할 수 있음을 보였다. 향후 국내뿐만 아니라 국외의 사이트를 포함한 동적 그리드 컴퓨팅과 보다 정확한 성능지수, 이질 적인 운영체제와 이질적인 프로세스 구조를 갖는 환경으로 확장하기 위해 서로 다른 운영체제 사이의 상대적인 성능지수, 서로 다른 프로세스 구조를 갖는 시스템들의 상대적인 성능 지수 등의 새로운 파라미터를 고려한 추가의 연구가 수행될 필요가 있다.

참 고 문 헌

[1] NEESgrid, http://it.nees.org

[2] The Grid Physics Network, http://www.griphyn.org

[3] K*Grid, http://www.gridcenter.or.kr

[4] Nicholas T. Karonis, Brian Toonen, Ian Foster, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface," In Proceedings of ASCM/IEEE SC'98 Conference, ACM press, 1998.

[5] Garbrielle Allen, Werner Benger, Thomas Dramlitsch, Tom Goodale, Hans-Christian Hege, Gerd Lanfermann, Andre Merzky, Thomas Radke, Edward Seidel, John Shalf, "Cactus Tools for Grid Applications," Cluster Computing, Vol.4(3), pp. 179-188, 2001.

[6] Cactus, An open source problem solving environment, http://www.cactuscode.org

[7] Garbrielle Allen, Thomas Dramlitsch, Ian Foster, Nicholas T. Karonis, Matei Ripeanu, Edward Seidel, Brian Toonen, "Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus," SC2001 November 2001, Denver.

[8] Garbrielle Allen, David Angulo, Ian Foster, Gerd Lanfermann, Chuang Liu, Thomas Radke, Ed Seidel, John Shalf, "The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Envrionment," The International Journal of High-Performance Computing Applications and Supercomputing 15(4), Winter, 2001

[9] Sathish S. Vadhiyar, Jack J. Dongarra, "Self Adaptivity in Grid Computing," CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE, Concurrency Computat. : Pract. Exper. 2004; 00:1-26.

[10] Carsten Ernemann, Volker Hamscher, Achim Streit, Ramin Yahyapour, "Enhanced Algorithms for Multi-Site Scheduling," In 3rd Int'l Workshop on Grid Computing, pp. 219-231, 2002.

[11] Chuang Liu, Lingyun Yang, Ian Foster, Dave Angulo, "Design and Evaluation of a Resource Selection Framework for Grid Applications," In Proceedings of the 11th IEEE Symposium oh High-Performance Distributed Computing, 2002.

[12] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of High Performance Computing Applications, 15(3). 200-222.

[13] I. Foster and C. Kesselman. editors. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, 1999.

[14] P. Roe, C. Szyperski, "Transplanting in Gardens: Efficient Heterogeneous Task Migration for Fully Inverted Software Architectures", Proceedings of the Fourth Australasian Computer Architecture Conference, Auckland, New Zealand, January 18-21, 1999.

[15] Gabrielle Allen, Tom Goodale, Michael Russell, Edward Seidel and John Shalf, "Classifying and enabling grid applications," CONCURRENCY-PRACTICE AND EXPERIENCE, Concurrency: Pract. Exper. 2000; 00:1-7.

[16] Sriram Krishnan, Dennis Gannon, "Checkpoint and Restart for Distributed Components in XCAT3," In Proceedings of the fifth IEEE/ACM International Workshop on Grid Computing, pp. 281~288, Pittsburgh, Pennsylvania, 8 November, 2004.
[17] I. Foster and Carl Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," International Journal of Supercomputing Applications, 11(2), 1997.
[18] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. "A directory service for configuring high-performance distributed computations," In Proc. 6th IEEE Symp. on High Performance Distributed Computing, pages 365-375. IEEE Computer Society Press, 1997.
[19] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W.Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," In The 4th Workshop on Job Scheduling Strategies for Parallel Processing," 1998.
[20] Karl Czajkowski, Ian Foster, and Carl Kesselman, "Co-allocation services for computational grids," In Proc. 8th IEEE Symp. on High Performance Distributed Computing. IEEE Computer Society Press, 1999.
[21] Joseph Bester, Ian Foster, Carl Kesselman, Jean Tedesco, and Steven, Tuecke, "GASS: A data movement and access service for wide area computing systems," In Proc. IOPADS'99. ACM Press, 1999.
[22] D. Abramson, K. Power, L. Kolter, "High performance parametric modelling with Nimrod/G: A killer application for the global Grid", In Proceedings of the International Parallel and Distributed Processing Symposium, Cancun, Mexico, 2000, pp. 520-528.
[23] Triana, An open source problem solving environment, http://www.triana.co.kr
[24] Kyu Hong Kim, Chongam Kim and Oh Hyun Rho, "Accurate Computations of Hypersonic Flows Using AUSMPW+ Scheme and Shock-aligned-grid Technique," AIAA 98-2422, 1998.
[25] Yoon, S., and Jameson, A., "Lower-Upper SymmerticGauss-Seidel Method for the Euler and Navier-Stokes Equations," AIAA Journla, Vol 26, No. 9, pp. 1025-1026, 1988.

김 영 균
1996년 금오공과대학교 전자계산기공학과 학사. 1998년 금오공과대학교 컴퓨터공학과 석사. 1999년~현재 금오공과대학교 전자공학과 컴퓨터공학전공 박사과정. 관심분야는 그리드 컴퓨팅, 클러스터 컴퓨팅, 패턴인식

오 길 호
1980년 서울대학교 전기공학과 학사 1982년 서울대학교 컴퓨터공학과 석사 1992년 University of Florida 컴퓨터공학과 박사. 1983년~현재 금오공과대학교 컴퓨터공학부 교수. 관심분야는 병렬/분산 소프트웨어공학, 실시간 시스템, 객체지향시스템

조 금 원
1993년 인하대학교 항공우주공학과 학사. 1995년 한국과학기술원 항공우주공학과 석사. 2000년 한국과학기술원 항공우주공학과 박사. 2000년~현재 한국과학기술정보연구원(KISTI) 선임연구원 관심분야는 e-Science, 그리드 컴퓨팅

고 순 홈
2000년 서울대학교 기계항공공학부 학사. 2002년 서울대학교 기계항공공학부 석사. 2002년~현재 서울대학교 기계항공공학부 박사과정. 관심분야는 CFD, 그리드 컴퓨팅, e-Science