Cactus와 Globus에 기반한 그리드 컴퓨팅 환경에서의 응용프로그램 수준의 체크포인팅을 사용한 동적 재구성 기법

(A Dynamic Reconfiguration Method using Application-level Checkpointing in a Grid Computing Environment with Cactus and Globus)

김 영 균 [†] 오 길 호 [††] 조 금 원 [†††] 나 정 수 [†††]
(Young Gyun Kim) (Gil-ho Oh) (Kum Won Cho) (Jeoung-Su Na)

요 약 본 논문에서는 Cactus와 Globus 기반의 그리드 컴퓨팅 환경에서 응용프로그램 수준의 체크포인팅을 사용한 동적 재구성(Dynamic Reconfiguration) 기법을 새로이 제안하였다. 기존의 동적 재구성은 특정 하드웨어와 운영체제에 종속적이었으나 제안한 방법은 특정 하드웨어와 운영체제의 지원 없이 동적재구성이 가능하고 응용프로그램도 동적 재구성을 고려할 필요 없이 프로그래밍이 가능하다. 제안한 동적 재구성 기법에서 초기 연산자원의 구성을 갖고 실행되는 작업은 실행 중에 동적으로 발견된 새로운 연산자원을 포함하여 계속 연산을 수행한다. 본 연구에서 제안된 방법은 새롭게 발견된 연산자원의 프로세서 성능과 유휴 메모리를 고려하여, 해당 연산자원을 수행중인 연산에 포함할 것인가 여부를 결정한다. 연산 중 연산 결과의 실시간 가시화를 가능하도록 하고 운영체제에 종속적이지 않은 응용프로그램 수준의 체크포인팅 기법을 사용하여 중간 연산결과를 저장한다. 새롭게 발견된 유휴사이트, 유휴프로세서를 포함하도록 연산자원의 구성을 재구성한 후 체크 포인팅 파일을 사용하여 작업을 계속 실행한다. 제안한 동적 재구성 기법은 K*Grid 환경에서 연산시간을 단축함을 확인하였다.

키워드 : 재구성, 그리드 컴퓨팅, 체크포인팅

Abstract In this paper, we propose a new dynamic reconfiguration method using application-level checkpointing in a grid computing environment with Cactus and Globus. The existing dynamic reconfiguration methods have been dependent on a specific hardware and operating system. But the proposed method performs a dynamic reconfiguration without supporting specific hardwares and operating systems and, an application is programmed without considering a dynamic reconfiguration. In the proposed method, the job starts with an initial configuration of computing resources and the job restarts including new resources dynamically found at run-time. The proposed method determines whether to include the newly found idle sites by considering processor performance and available memory of the sites. Our method writes the intermediate results of the job on the disks using system-independent application-level checkpointing for real-time visualization during the job runs. After reconfiguring idle sites and idle processors newly found, the job resumes using checkpointing files. The proposed dynamic reconfiguration method is proved to be valid by decreasing total execution time in K*Grid.

Key words : Reconfiguration, Grid computing, Checkpointing

† 학생회원 : 금오공과대학교 전자공학과
ygkim2004@paran.com
†† 종신회원 : 금오공과대학교 컴퓨터공학부 교수
gilho@kumoh.ac.kr
††† 비 회 원 : 한국과학기술정보연구원 슈퍼컴퓨팅응용실 연구원
ckw@kisti.re.kr
ninteger@kisti.re.kr
논문접수 : 2005년 3월 14일
심사완료 : 2005년 8월 22일

1. 서 론

최근 그리드 컴퓨팅이라는 분야가 새롭게 각광받고 있다. 그리드 컴퓨팅은 소형 및 대형 컴퓨터들과 PDA, 파일 서버들과 그래픽 장치들을 포함하는 다양한 컴퓨팅 자원들의 집합을 고성능 네트워크에 연결하여 공동 활용하는 것으로 정의할 수 있다. 다양한 컴퓨팅 자원들의 집합을 연결하는 네트워크 기술은 고속의 ATM으로

부터 무선 또는 모뎀 접속 등의 어떤 것도 가능하다. 이들 연결된 자원들을 활용하는 것은 단일의 컴퓨터로는 불가능했던 대규모의 시뮬레이션을 가능하게 하고 지리적으로 분산된 협동작업(Collaborations)의 연산을 지원하며 자원들의 원격 사용을 단순하게 한다[1].

그러나 기존의 단일 프로세서 시스템들(Single-processor systems)보다 병렬 및 분산 시스템들의 자원 관리가 훨씬 어렵다는 것이 문제이다. 특히 그리드 컴퓨팅과 같이 수많은 컴퓨팅 자원들이 네트워크를 통해 연결될 경우는 더욱 그러하다. 기존의 정적인 자원 관리 전략들은 자원들이 정적으로 할당되고 작업은 실행동안 단일의 자원 스케쥴링 단위로 다루어진다. 이러한 정적인 할당 방법은 그리드 컴퓨팅과 같이 자원이 동적으로 변화하는 환경에서는 유휴 자원들을 효과적으로 사용할 수 없다. 그리드 컴퓨팅과 같은 동적인 분산 처리 환경에서는 시스템의 요구가 변할 때 동적으로 자원들의 할당을 조절할 수 있는 방법이 보다 효과적이다. 그리드 컴퓨팅 환경에서 보다 효과적인 자원 관리 방법은 자원들의 요구 또는 시스템의 작업부하(Workload)에 대한 변화에 재빠르게 반응하는 방식이다[2]. 이렇게 하기 위해 본 연구에서는 Cactus 프레임웍에서 CFD(Computational Fluid Dynamics) 응용 프로그램의 실행 도중에 더 이상 사용되지 않는 유휴 프로세서들이 추가로 발생할 경우 현재의 Cactus 프레임웍에서 CFD 응용 프로그램을 좀 더 빠르게 실행 할 수 있도록 유휴 프로세서들을 추가한다. 이것은 Cactus CFD 응용 프로그램의 파라미터 파일과 글로버스 RSL(Resource Specification Language) 파일의 변경을 요구 한다. 이후 실행중인 Cactus CFD 응용 프로그램의 현재 상태를 응용프로그램 수준의 체크 포인팅 기법을 사용하여 하드디스크로 수록하고 현재 연산을 중단한다. 다음으로 유휴 사이트의 유휴 프로세서를 추가하도록 파라미터 파일과, RSL 파일을 변경한 후, 수정된 RSL 파일과 함께 체크 포인팅 파일에 기반 하여 Cactus CFD 응용프로그램을 중단된 지점부터 재실행한다. 결과적으로 제안한 기법은 새롭게 발견된 연산 자원들을 포함하여 연산을 수행함으로써 Cactus CFD 응용프로그램의 동적인 재구성(Reconfiguration)을 지원 한다. 제안한 방법은 K*Grid 환경에서 실험을 통해 연산시간을 단축함을 검증하였다.

2. 관련 연구

2.1 그리드 컴퓨팅과 재구성

그리드(Grid) 컴퓨팅은 지리적으로 분산된 다중의 사이트에 구축된 HPC(High-Performance Computer)와 클러스터 컴퓨터가 초고속 인터넷으로 연결되어 대용량의 연산자원들을 요구하는 많은 응용들에 대해 연산자원들을 제공하는 것을 목표로 한다. 단일 사이트의 HPC, 클러스터 시스템이 제공하지 못하는 대용량, 고속의 연산자원들을 제공하여 지구규모의 기상예측, 블랙홀의 충돌, 초신성의 폭발과 같은 천문학적 문제, 우주선, 항공기 등의 유체 역학 문제, 유전자 분석 등의 계산에 널리 사용이 되고 있다[3,4].

그리드 컴퓨팅과 같은 동적인 분산 처리 환경에서는 시스템의 요구가 변할 때 동적으로 자원들의 할당을 조절할 수 있는 방법이 보다 효과적이다. 따라서 그리드 컴퓨팅 자원들의 할당을 동적으로 조절하여 응용 프로그램의 수행을 지원하도록 본 논문에서는 재구성 가능한 컴퓨팅(Reconfigurable computing)의 개념을 도입한다.

재구성 가능한 컴퓨팅 또는 재구성 가능한 구조(Reconfigurable architectures)는 FPGAs(Field Programmable Gate Arrays)[5]로부터 발전되어 왔다. FPGAs는 일련의 논리 블록(Logic blocks)과 상호 접속 네트워크(Interconnection network)로 구성되어 있다. 논리 블록들의 기능과 상호 접속 네트워크에의 접속은 하드웨어로 구성 데이타 비트들을 다운로딩 함으로서 수정되어질 수 있다[6].

그림 1은 이러한 FPGAs에 기반한 전통적인 정적으로 구성 가능한 컴퓨팅의 개념을 보여 주고 있다. 하드웨어가 한번 구성된 후 연산을 실행한다. 재구성 가능한 컴퓨팅의 개념은 하드웨어보다 훨씬 유연성(Flexibility)이 뛰어나고 소프트웨어보다 뛰어난 성능을 제공하기 위해 하드웨어와 소프트웨어간의 차이를 극복하기 위한 패러다임(Paradigm)으로서 등장했다[1]. 이러한 재구성 가능한 컴퓨팅은 성능상의 타협 없이 훨씬 더 큰 유연성을 제공하기 위해 실행시간에 적응할 수 있는 하드웨어를 사용하는 것이다[7].

그림 2는 실행시간에 적응할 수 있는 하드웨어를 사용하는 동적 구성 가능한 컴퓨팅의 개념을 보여주고 있다. 한번 계산이 이루어진 후 하드웨어를 재구성함으로서 연산자원을 재사용한다. 로직(Logic)과 상호접속망(Interconnection network)의 구성은 실행동안에 즉시 적응한다. 이 접근 방식은 연산 작업의 특성에 하드웨어를 밀접하게 변경함으로서 기존의 접근방식보다 좀더 고성능을 달성할 수 있다.

그림 3은 실행을 위해 요구되는 하드웨어의 구성이 구조의 특성과 모델 기반의 분석을 통해 생성되는 모델 기반의 재구성의 과정을 보여주고 있다. 생성된 구성은 하드웨어를 재구성하기 위해 사용되고 연산을 실행한다.

입력 데이타 집합 또는 외부 환경에 독립되어 실행 작업들(Executing tasks)의 개수가 고정된 개수만을 가지는 응용프로그램을 정적으로 구성된 병렬 응용프로그램(Statically configured parallel application)이라고 한

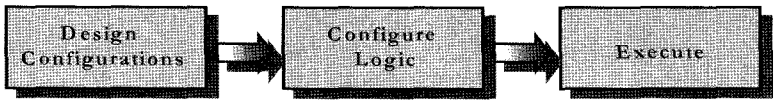

그림 1 전통적인 정적 구성 가능한 컴퓨팅(Static configurable computing)

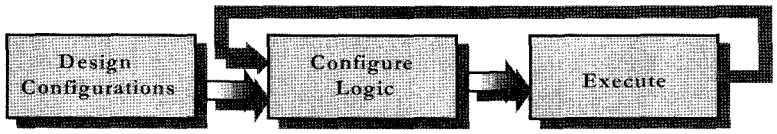

그림 2 전통적인 동적 구성 가능한 컴퓨팅(Dynamic configurable computing)

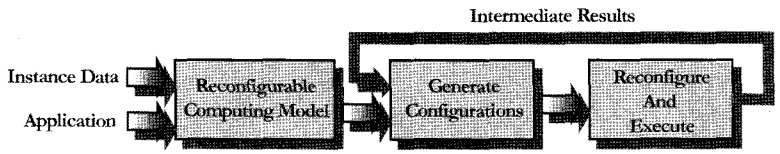

그림 3 모델 기반의 계산과 재구성

다. 반면 입력 데이타 집합에 따라 실행되는 작업들의 개수가 달라지는 응용프로그램을 동적으로 구성 가능한 응용프로그램(Dynamically configurable application)이라고 한다. 또한 동일한 입력 데이타 집합에 대해 하나 이상의 태스크들(Tasks)을 실행할 수 있는 응용프로그램도 동적으로 구성가능하다고 한다.

재구성 가능한 병렬 응용프로그램들(Reconfigurable parallel application)은 다음의 속성 중 하나 이상의 특성을 갖는다.

- 프로그램의 실행 중 태스크들의 개수를 변경할 수 있는 능력
- 태스크들의 실행 유닛들(Execution units)로 매핑을 변경할 수 있는 능력
- 실행 중에 데이타 요소들과 태스크들의 연결성(Affinity)을 변경할 있는 능력
- 실행 중에 데이타들의 유형을 다른 유형으로 전환할 수 있는 능력

응용프로그램이 위 속성 중 오직 하나만을 가질 때 동적으로 재구성가능(Dynamically reconfigurable)하다고 할 수 없다[2].

소프트웨어를 재구성할 수 있는 능력은 환경의 변화에 적응하는 응용프로그램들의 구현, 온라인 소프트웨어 업그레이드, 그리고 응용프로그램의 기본 기능에 제공되지 않는 추가적인 기능들로 확장하는 것을 포함한 다양한 이유로 유용하다. 재구성 능력(Reconfigurability)은 자율적인 시스템들(Autonomic systems)이 그들의 환경 변화에 적응할 수 있도록 하고, 관찰된 실행 프로파일(Execution profile)에 기초하여 시스템의 기능을 동적으로 최적화 하는데 유용하며, 인간의 개입 없이 에러들과 결함들로부터 복구하기 위한 기본적인 능력을 제공한다[8].

프로그램이 재구성될 때, 프로그램 상태의 변화가 있다. 그러한 상태의 변화가 재구성 지점(RP; Reconfiguration Point) 이전의 프로그램의 의미(Semantics)를 변경해서는 안 된다. 또한, RP 이후의 예기치 않은 프로그램의 행동을 유발해서도 아니 된다. 일반적으로 재구성은 다음과 같은 규칙을 따라야 한다.

1. 응용프로그램의 의미는 RP로부터 변경되어서는 안 된다.
2. RP이전에 프로그램의 상태를 정의한 모든 데이타는 RP로부터 보존되어야 한다.
3. RP에서 중단된 어떤 작업이 있다면, 그 작업과 관련된 종료되지 않은 모든 계산들은 RP이후에 존재하는 1개 이상의 작업들에 할당되어야 한다.
4. 재구성을 따르는 알고리즘의 변화가 있다면, 이전의 알고리즘은 기대된 프로그램의 행동이 유지되는 동등한 새로운 알고리즘으로 대체되어야 한다.
5. 재구성은 병렬성의 정도(Degree of parallelism)의 변화의 결과로 데드락(Deadlock)이나 라이브락(Livelock)을 발생시켜서는 안 된다.

위의 규칙들을 따르는 재구성 가능한 응용프로그램들(Reconfigurable applications)은 타당한 재구성 가능 응용프로그램들(Valid reconfigurable applications)이라고 한다[2].

소프트웨어와 알고리즘에 재구성 가능한 컴퓨팅을 적용하는 많은 연구들이 있다[1,9-11]. 이러한 대표적인 연구로 Luís Fabrício는 시스템의 성능을 최대화하기 위해 병렬 작업 스케쥴링 알고리즘(Gang scheduling)에 재구성 가능한 컴퓨팅의 개념을 적용하여 RGSA (Reconfigurable Gang Scheduling Algorithm)를 제안

했다[1]. RGSA는 성능측정지수(utilization, mean jobs response time 등)와 작업부하 특성들(mean jobs parallelism degree 등)과 같은 파라미터 값들의 입력에 따라서 스스로 자신을 재구성한다. J. Jann은 시스템의 재부팅 없이 논리적으로 분할된 서버들 상에서 하드웨어 자원들(프로세서들, 메모리, 그리고 I/O 슬롯들과 같은 자원들)의 추가와 제거가 가능하도록 분할된 파티션들의 사용량을 감시하는 자율적인 에이전트(Autonomic agent)를 도입하여 동적 재구성을 지원 하는 연구를 수행 하였다[12].

실행중인 작업을 동적 재구성하기 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 또는 Application-level checkpointing)으로 나누어진다. 시스템 수준의 체크포인팅은 운영체제나 미들웨어에 의해 응용프로그램에 투명하게 자동적으로 제공되는 기법이다. 이것은 응용프로그램의 완전한 프로세스 이미지(Process image)를 캡쳐 하는 것이다. 사용자 정의 체크포인팅은 응용프로그램의 상태에 대한 캡쳐를 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[13].

동적인 그리드 컴퓨팅에서 Cactus의 연산 작업을 이주(Migration)시키는 많은 연구들이 선행되었다. 대표적인 연구로서 Garbrielle Allen은 Cactus의 연산 작업을 응용프로그램 수준의 체크포인팅기법을 적용하여 글로버스 툴킷 기반의 그리드 환경에서 Cactus 응용프로그램의 작업이주를 시도하였다[14]. 그러나 이 방법은 연산자원에 대한 평가와 재구성에 관한 고려 없이 단순 작업 이주 방법만을 적용하였다. Sathish S. Vadhiyar는 GrADS 시스템에서 자원들의 부하 증가와 감소, 응용 프로그램이 실행되는 시점에서 시스템의 부하, 작업 이주로 얻어지는 성능상의 이득을 고려하였다[15]. GrADS 또한 성능상의 이득은 고려하였으나 재구성에 대한 고려가 없는 동적 그리드 컴퓨팅 기법을 사용하였다. Cartsten Ernemann은 계산 그리드(Computational Grid)상에서 단일 작업의 자원 소모를 다중 사이트 스케줄링에 사용하는 적응형 다중 사이트 스케줄링(Adaptive Multi-site Scheduling)기법을 제안하였다[16]. 이 방법은 단일 사이트 보다 다수 개의 사이트에서 고정된 개수의 노드를 병렬로 사용하여 연산을 수행하는 방법으로 연산을 수행하는 노드의 개수를 작업 제출시 명시적으로 요구하는 문제점이 있다. Chuang Liu는 Cactus 응용 프로그램에 대해 CPU의 부하, 메모리 크기, 분할된 작업들의 통신을 고려하여 다중 사이트의 자원 할당에 대해 연구하였다[17]. 이 방법은 다중 사이트로 자원 할당을 확장하였으나 작업 할당 후 새롭게 발견된 연산 자원을 활용할 수 없다는 문제점을 갖고 있다. 기존의 그리드 컴퓨팅의 연구는 대부분 작업이주와 작업 제출시 연산 노드 개수를 명시하는 정적 재구성 컴퓨팅만을 고려하였으며 동적 재구성 컴퓨팅에 대한 연구는 이루어지지 않았다. 본 연구에서는 이러한 재구성 가능한 컴퓨팅의 개념을 자원들이 동적으로 변화하는 그리드 컴퓨팅 환경에 적용한다. 그리드 컴퓨팅 환경에서 Cactus CFD 응용프로그램의 수행 중 유휴 자원(유휴 사이트, 유휴 프로세서들)이 발견 되면 현재 연산 작업에 유휴 자원들을 추가 할당하여 연산을 보다 빨리 수행하도록 실행 환경을 재구성 한다.

2.2 Cactus와 Globus를 갖는 그리드 컴퓨팅 환경

2.2.1 Cactus의 특성 및 구조

Einstein 방정식을 계산하기 위한 프레임웍으로 개발된 Cactus는 과학자와 공학자들을 위한 단일화된 모듈 방식과 병렬 계산 프레임웍을 제공하는 일반적인 목적의 오픈소스로서의 문제해결 환경(Problem solving environment)으로 발전되어 왔다[4]. Cactus라는 이름은 확장 가능한 인터페이스를 통해 중앙의 코어(Flesh)에 연결되는 응용프로그램 모듈들(Thorns)을 갖는 구조에서 유래 했다. Thorns은 데이타의 분산과 체크 포인팅과 같은 기능뿐만 아니라 전산유체역학과 같은 특정 과학 또는 공학 응용들에 맞추어 구현 된다. Cactus 툴킷 thorns의 확장 집합은 Globus 툴킷, HDF5 병렬 파일 I/O, PETSc 과학 연산 라이브러리, 웹 인터페이스 그리고 시각화 툴들과 같은 개발되어 온 많은 소프트웨어들에 대한 접근을 제공한다. Cactus는 Uniprocessor, Cluster, Supercomputer를 포함한 많은 구조들에서 실행된다. 병렬성과 이식성은 병렬 MPI 드라이버 계층, I/O시스템 등의 존재여부와 그 특성을 숨김으로서 달성

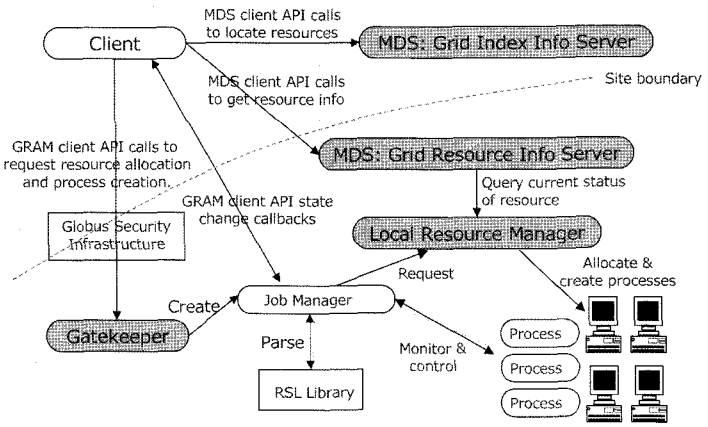

그림 4.GRAM의 주요 요소들

```
+                                                    // Indicate start of RSL script
( &(resourceManagerContact="venus.gridcenter.or.kr")  // where to run the job
   (count=4)                                         // this selects 4 CPUs
   (label="subjob 0")                                // convenient job name
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)        // first subjob
                (LD_LIBRARY_PATH /usr/local/grid/GLOBUS/lib/))
   (directory="/home/kgrid/kgrid021/reconfiguration/")   //where to find the exe
   (executable="/home/kgrid/kgrid021/reconfiguration/cactus_m9")
                                                     //what to run
   (arguments="Euler3D2.par")                        //which argument
)
```

그림 5 RSL 파일의 형식

된다. 이들 계층은 thorns으로 구현된다. Grid-enabled Cactus는 MPI를 이질적인 컴퓨터들의 집합에 활용할 수 있도록 설계 및 구현된 MPICH-G를 사용한다. MPICH-G는 자원 발견, 인증, 자원 할당, 실행 프로그램의 실행, 관리, 제어들을 위해 Globus 서비스를 사용한다.

2.2.2 Globus 툴킷

글로버스 툴킷은 그리드와 그리드 응용들을 지원하는 서비스들과 소프트웨어 라이브러리들의 집합이다. 이 툴킷은 보안, 정보, 자원 관리, 데이타 관리, 통신, 장애 감지, 그리고 이식성을 제공하는 소프트웨어들을 포함하고 있다. 유용한 그리드 응용프로그램들과 프로그래밍 툴들을 개발하기 위해 함께 또는 독립적으로 사용될 수 있는 콤포넌트들의 집합으로 패키지화 되어 있다. 글로버스 툴킷 콤포넌트들은 단일 로그인으로 그리드 자원들에 접근할 수 있도록 인증하는 GSI(Grid Security Infrastructure), 원격 자원 할당과 프로세스 생성, 모니터링, 그리고 관리 서비스를 제공하는 GRAM(Grid Resource Access and Management) 프로토콜과 서비스, 복제된 데이타 집합들의 위치들, 서버 구성, 네트워크 상태와 같은 상태 정보와 시스템 구성의 발견 및 접근을 단일의 프레임웍으로 제공하는 확장 가능한 그리드 정보 서비스를 제공하는 MDS(Metacomputing Directory Service), 고속의 데이타 전송 프로토콜인 GridFTP를 포함한다. 다양한 상위계층의 서비스들이 이들 기본적인 콤포넌트들을 사용하여 구현된다.

GRAM의 주요 구성요소들은 그림 4와 같으며, GRAM 클라이언트 라이브러리, 게이트키퍼, RSL 파싱 라이브러리, 잡 매니저, GRAM 리포터로 구성되어 있다. 인증을 위해 GSI를 사용한다[18]. GRAM은 다음에 대해 책임을 진다.

1. 자원 요청(Resource requests)을 나타내는 RSL Specifications를 처리하고, 요청을 거부하거나, 요청을 만족하는 1개 이상의 프로세스들(하나의 작업)을 생성한다.
2. 자원 요청에 의해 생성된 작업들의 원격 모니터링과 관리를 가능하게 한다.
3. 관리하는 자원들의 현재 상태에 관해 MDS 정보 서비스를 주기적으로 갱신한다.

RSL 파일은 그림 5와 같은 형식으로 구성되어 있으며 글로버스 작업을 수행하기 위한 클러스터의 프로세

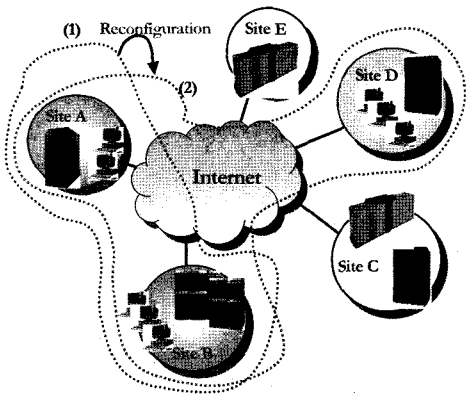

그림 6 그리드 컴퓨팅 환경에서의 동적 재구성을 통한 연산자원의 동적확장

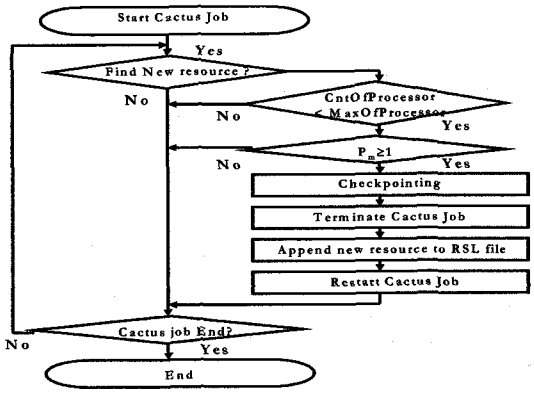

그림 7 체크포인팅 기법을 사용한 동적 재구성 지원 순서

서 개수, 프로그램 실행 디렉토리, 라이브러리, 실행에 필요한 파라미터 파일 등의 컴퓨팅 연산 자원들의 위치 및 명칭을 지정해 준다.

3. 제안한 동적 확장성

제안한 동적 확장 기법은 그림 6의 (1)과 같이 연산 작업의 초기에 발견된 Site A와 Site B의 자원을 가지고 연산을 시작한다. 시간이 경과한 후, Site D의 자원이 유휴 상태로 변경되어 유휴 자원으로 발견되면 Site A와 Site B에서 수행중인 작업을 위해 동적으로 연산에 참여할 수 있도록 확장 된다. 이렇게 하기 위해 Site A와 B에서 수행 중인 작업을 체크 포인팅을 사용하여 저장한 후 잠시 중단한다. 새로 발견된 유휴 자원 Site D를 Site A와 Site B의 자원을 사용하여 수행 중인 작업에 포함 시킬 것인가 여부를 식 (1)을 사용하여 P_m의 값을 평가한 후 $P_m \geq 1$ 자원에 대해 포함 시킬 것을 결정 한다. Site D가 조건 $P_m \geq 1$를 만족하면, 그림 6의 (2)와 같이 결정된 Site D의 자원을 포함하도록 RSL 파일을 수정한 후, 체크 포인팅 파일을 사용하여

```
+                                          //Indicate start of RSL script
( &(resourceManagerContact="Site A")       //where to run the job
   (count=1)                               //this selects 1 CPUs
   .....
)
( &(resourceManagerContact="Site B")       //where to run the job
   (count=2)                               //this selects 2 CPUs
   .....
)
```

그림 8 동적 재구성 이전의 RSL 파일

```
+                                          //Indicate start of RSL script
( &(resourceManagerContact="Site A")       //where to run the job
   (count=1)                               //this selects 1 CPUs
 .....
)
( &(resourceManagerContact="Site B")       //where to run the job
   (count=2)                               //this selects 2 CPUs
 .....
)
( &(resourceManagerContact="Site D")       //where to run the job
   (count=16)                              //this selects 16 CPUs
 .....
)
```

그림 9 동적 재구성 이후의 RSL 파일

중단된 작업을 재개 한다.

제안한 동적 재구성 과정을 간략히 순서도로 나타내면 그림 7과 같다. 이와 같이 초기에 적은 연산 자원을 가지고 실행된 작업이 수행 중 새롭게 유휴 상태로 발견된 자원을 포함하여 연산을 수행하고 전체 연산 작업의 시간을 단축할 수 있도록 RSL 파일을 변경함으로서 동적으로 재구성된다.

$P_m < 1$인 사이트의 연산 자원을 추가하는 것은 프로세서간의 동기화를 요구하는 연산 작업의 경우 전체 연산 시간을 증가시킬 수 있기 때문에 동적 재구성에서 제외 한다. 그림 8은 재구성 이전의 연산에 참여하는 RSL 파일을 나타내고, 그림 9는 새롭게 발견된 연산자원이 추가된 재구성 이후의 연산에 참여하는 RSL 파일을 나타낸다.

클락 주파수당 1개의 명령어(operations)를 처리하는 부하가 없는 이상적인 단일의 프로세서를 갖는 시스템의 초당 처리 가능한 오퍼레이션의 수는 클락 주파수 X 1 이다. 이러한 프로세서가 N개가 존재할 때 처리 가능한 명령어의 개수는 클락 주파수 X N이다. 0~1사이로 정규화된 부하가 있는 경우, 부하 값만큼 명령어를 처리 할 수 없으므로 (1-부하 값)X클락 주파수 X N개의 오퍼레이션을 초당 처리 가능하다. 클락 주파수가 1Ghz이고, 부하 값이 0.3인 프로세서가 4개 있다면, 최대 (1-0.3)X1GX4개의 오퍼레이션을 처리 할 수 있다. 32Bits 프로세서를 기준으로 하면, 32Bits로 나눈 값을 곱해 준다. 즉 이론적으로 최대 (1-0.3)X1GX4X1개 이하의 32Bits 오퍼레이션을 처리 할 수 있다. 64Bits 프로세서는 32Bits 프로세서보다 이론적으로 최대 (1-0.3)X1GX4X2개 이하의 오퍼레이션을 처리 할 수 있다. 이 개념을 각 사이트마다 적용하여 각 사이트의 이론상의 초당 최대 처리 가능한 오퍼레이션의 수를 구하고 이 수의 비를 구하면 개략적인 성능 비를 구하는 것이 가능하다. 본 논문에서는 식 (1)과 같이 정량화 하여 상대적인 성능지수를 구한다.

제안한 식 (1)은 프로세서의 초당 처리할 수 있는 이론적인 최대 오퍼레이션 수에 기반하고 있기 때문에 상대적인 성능지수 P_m이 1보다 큰 값을 갖는 다는 것은 현재 사이트의 연산자원 보다 초당 처리할 수 있는 이론적인 최대 오퍼레이션이 많다는 것을 의미한다. 따라서 P_m이 1보다 큰 값을 갖는 것이 1보다 작은 값을 갖는 경우보다 유리하다.

재구성 사이트와 현재 사이트와의 상대적인 성능 지수 P_m은, 현재 노드의 현시점에서의 평균 프로세서 부하를 L_c, CPU 속도를 $S_{c,cpu}$, 유휴 CPU 개수를 $N_{c,cpu}$라 하고, 재구성할 사이트의 현 시점에서의 평균 프로세서 부하를 L_m, CPU속도를 $S_{m,cpu}$, 유휴 CPU개수를 $N_{m,cpu}$이라고 하면, 현재 사이트에서 사용할 수 있는 연산 능력은 부하를 제외한 값이므로 $(1-L_c)$ 이고, 재구성할 사이트에서의 가용할 수 있는 연산 능력도 부하를 제외하면 $(1-L_m)$이 된다.

$L_{c,free}=(1-L_c)$, $L_{m,free}=(1-L_m)$이라 할 때, $L_{m,free}=0$라는 것은 재구성 할 사이트에 연산 능력에 여유가 없다는 것을 뜻한다. $L_{c,free}=0$이라는 것은 현재 노드가 과부하 상태라는 의미를 갖는다. 그리고 현재 노드의 유휴 주기억 용량을 $M_{c,freememory}$, 보조기억장치의 용량을 $S_{c,freedisk}$, 재구성할 노드의 유휴 주기억 용량을 $M_{m,freememory}$, 보조기억장치의 용량을 $S_{m,freedisk}$이라 하자, 이때 클러스터 시스템 내의 프로세서들 사이를 연결한 네트워크의 통신 속도가 빠를수록 재구성 했을 때 이득이 크다. 또한 연산에 참여 가능한 가용 프로세서의 비트수가 클수록 재구성의 이득이 크다.

현재 사이트의 클러스터 시스템의 가용 프로세서를 연결하는 네트워크의 전송 속도를 $S_{c,connection}$, 재구성할 사이트의 클러스터 시스템의 가용 프로세서의 전송 속도를 $S_{m,connection}$이라하고, 재구성할 사이트의 가용 프로세서의 비트수를 $B_{m,cpu}$, 현재 사이트의 가용 프로세서의 비트수를 $B_{c,cpu}$라 하자. α,β,γ는 가중치로서, $0 \le \alpha,\beta,\gamma \le 1$이라 할 때, 가용 가능한 연산 능력의 상대적인 성능 지수 P_m은 식 (1)과 같이 계산된다.(단, $0 \le L_m, L_c \le 1$)

$$P_m = \alpha\left(\frac{L_{m,free}\times S_{m,cpu}\times N_{m,cpu}}{L_{c,free}\times S_{c,cpu}\times N_{c,cpu}}\times\frac{S_{m,connection}}{S_{c,connection}}\times\frac{B_{m,cpu}}{B_{c,cpu}}\right) + \beta\frac{M_{m,freememory}}{M_{c,freememory}}+\gamma\frac{S_{m,freedisk}}{S_{c,freedisk}}$$
$$\text{if}\quad L_{c,free}=0\ then\ \ P_m = L_{m,free}\times S_{m,cpu}\times N_{m,cpu}\times S_{m,connection}\times B_{m,cpu} \qquad (1)$$

식 (1)은 현재 사이트의 프로세서의 가용능력과 재구성할 사이트의 프로세서의 가용 능력을 프로세서의 부하, 유휴 주기억 용량, 보조기억장치 용량, 네트워크의 통신 속도를 고려하여 상대적인 비율로 나타낸 것이다.

α,β,γ는 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. 유휴 주기억 장치 용량과 보조 기억 장치의 용량이 성능지수 P_m에 미치는 영향이 극소한 경우에는 $\alpha = 1, \beta = \gamma = 0$로 놓을 수 있다.

P_m의 값에 따라서, $P_m < 1$ 경우, 현재 사이트보다 가용 연산 능력이 떨어지는 사이트와의 재구성이다. $P_m = 1$ 인 경우, 동일하거나 비슷한 연산 능력을 갖는 노드와의 재구성이며, $P_m > 1$인 경우, 현재 사이트보다 연산 능력이 우수한 사이트를 포함하는 재구성이다. 일반적으로 $P_m \ge 1$인 사이트로의 재구성을 가정한다. 동

```
Step 1:Repeat
Step 2: L ← { all of available sites in a grid computing }
Step 3: CntOfProcessor ← N_c,cpu
Step 4: For each m ∈ L Do
        Begin
Step 5:   Get current site and new finding site's CPU load value,
            available processors, free memory size(main memory and disk).
Step 6:   Compute P_m from Step 5.
Step 7:   CntOfProcessor = CntOfProcessor + N_m,cpu
Step 8:   If (P_m ≥ 1) And (CntOfProcesspr ≤ MaxOfProcessor) Then
Step 9:   Begin
            Checkpointing current job's state to files.
            Terminate current cactus job.
            Append site m into RSL file.
            Restarts the job from checkpoint files using new RSL file.
Step 10:  Else
            CntOfProcessor = CntOfProcessor - N_m,cpu
Step 11:  End if
Step 12: End for
Step 13: Until CFD computation terminates.
```

그림 10 제안한 동적 재구성 알고리즘

적 재구성으로 인한 과도한 연산자원의 확장으로 다른 연산 작업들이 프로세서를 할당 받을 수 있는 연산자원이 고갈된 스타베이션(Starvation)을 방지하기 위해 연산에 참여하는 전체 프로세서 개수의 최대 값을 제한하는 방법을 사용할 수 있다. 본 논문에서는 연산에 참여하는 프로세서 개수의 최대 값을 제한하는 방법을 사용한다. 식 (1)을 사용한 동적 재구성 알고리즘은 그림 10과 같다. 그림 10에서 L은 그리드 컴퓨팅에 참여하는 유효한 사이트들의 집합이다.

그림 10의 Step 5에서의 부하 값은 Linux 운영체제가 제공하는 평균 부하 값을/proc/loadavg로부터 주기적으로 수집하여 정규화한 후 사용하였다. 프로세서의 정보 및 클락 속도는 pbsnodes -a와 /proc/cpuinfo, 메모리 사용 정보는 /proc/meminfo, 디스크 사용 정보는 df, 네트워크 통신 속도는 dmesg 명령어로부터 구한 값을 성능지수의 기준 값으로부터 나눈 값을 사용하였다. 이들 명령어로부터 값을 구하는 데 소요되는 시간이 추가로 요구된다. 제안한 방법은 응용프로그램 수준의 체크 포인팅 기법을 사용하여 RSL 파일의 수정을 통해 연산노드를 늘려가는 방식을 취하고 있기 때문에 동적으로 연산에 포함된 프로세서마다 새롭게 분할된 연산을 수행하는 프로세스가 생성되어 실행되므로 2.1절에서 설명한 재구성이 따라야 하는 규칙 1~5번까지 모두 만족한다.

4. 구현

제안한 방법은 그림 11에서 Reconfiguration Manager와 Reconfiguration Agent로 구성되었으며, Perl v5.6.1을 사용하여 Redhat Linux 7.3 운영체제에서 구현하였다. 실험을 하기 위해 사용한 3차원 Euler 방정식은 Fortran 77로서 Cactus 4.0 beta 12 프레임웍에서 구현되었다. Reconfiguration Manager는 연산에 참여하기로 결정된 사이트를 포함하는 RSL 파일을 기반으로 하여 작업을 중단하거나 재 시작 시키는 역할을 수행한다. 또한 Reconfiguration Agent로부터 수집된 정보를 바탕으로 연산에 참여할 수 있는 사이트인지를 결정한다. Reconfiguration Agent는 Agent가 위치한 사이트의 부하 정보, 시스템 관련 정보를 정기적으로 수집해서 Reconfiguration Manager로 전송하는 역할을 수행한

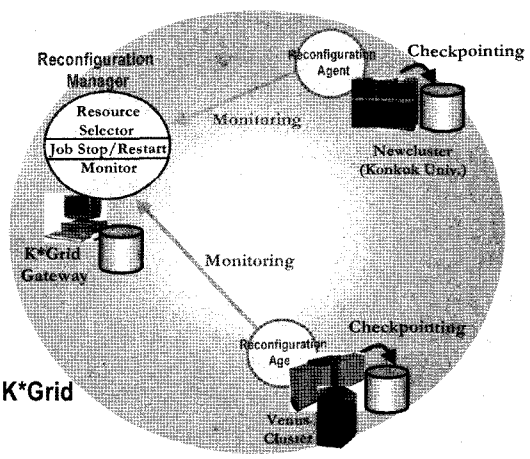

그림 11 동적 Reconfiguration Manager와 Agent

다. Reconfiguration Manager는 Resource Selector, Job Stop/Restart, Monitor 부분으로 구성되어 있으며, Resource Selector는 Reconfiguration Agent로부터 전송되어 온 값들을 바탕으로 재구성에 참여할 연산자원을 결정하는 역할을 수행하고, Job Stop/Restart는 Resource Selector가 결정한 연산자원을 포함한 RSL을 사용하여 체크포인팅 파일로부터 연산을 재개 하는 역할을 수행한다. Monitor는 각 사이트의 Reconfiguration Agent로부터의 값을 주기적으로 수집하고 Reconfiguration Agent의 수행을 감시하는 역할을 수행한다.

동적으로 발견되는 사이트의 프로세서는 표 1의 성능지수들과 같이 Reconfiguration Agent로부터 수집된 정보를 바탕으로 상대적인 연산능력을 나타내는 P_m 값을 계산한다. 그리고 식 (1)로 부터 구해진 P_m의 값을 바탕으로 동적확장에 포함될 것인지를 결정한다. 이후 연산에 포함할 것으로 결정된 연산자원은 RSL 파일에 추가됨으로써 연산에 참여하게 된다. 응용 프로그램 수준의 체크 포인팅 파일을 사용하여 현재 연산 작업의 상태를 저장한 후 중단된 작업은 새롭게 추가된 연산자원을 갖는 RSL 파일을 통해서 재실행을 함으로서 동적 재구성을 수행 한다.

컴퓨터의 정적인 성능 지수는 계산의 편리를 위해서 그리드 컴퓨팅에 참여할 사이트내의 프로세서의 성능지수를 기준값으로 나눈 값을 사용한다. 성능 지수의 기준 값으로서 CPU Clock속도 $S_{basis,cpu}$=1GHz, 주기억장치의 용량 $M_{basis,freememory}$=512Mbytes, 하드디스크 용량 $S_{basis,freedisk}$=40Gbytes, 가용프로세서를 연결하는 네트워크의 전송 속도 $S_{basis,connection}$=1Gbps, 가용 프로세서의 비트 수 $B_{basis,cpu}$=32Bits를 사용한다.

성능지수의 기준 값을 사용해서 KISTI Venus와 건국대 Newcluster 시스템의 성능지수를 계산하면 표 1과 같다.

표 1 K*Grid의 Venus, Newcluster 시스템의 기준값에 대한 성능지수

성능지수	KISTI Venus	KonKuk Univ. Newcluster
S_{cpu} (프로세서의 클락속도)	2.0	4.0(2.0×2)
N_{cpu} (가용 프로세서의 개수)	0~63	0~14
$M_{freememory}$ (유휴 메모리의 크기)	0~1.0	0~2.0
$S_{freedisk}$ (유휴 디스크의 크기)	0~1.0	0~0.4
$S_{connection}$ (네트워크의 전송 속도)	0.1	1.0
B_{cpu} (프로세서의 비트 수)	1.0	1.0

5. 실험환경 및 결과

실험 환경은 Globus 2.2.4와 MPICH-G2 1.2.5-1a를 사용하는 K*Grid에서 수행 하였다. 실험에 사용한 K*Grid의 KISTI Venus, 건국대학교의 Newcluster 시스템의 사양은 표 2와 같이 Linux Cluster로서 관리 노드를 제외하고 각각 63개와 14개의 프로세서로 구성되어 있다.

실험에 사용한 K*Grid의 Venus 클러스터 시스템은 그림 12와 같이 KISTI내에 위치하며 Newcluster는 KISTI와 약 140Km거리를 갖는 건국대에 위치한다. Venus와 Newcluster는 1Gbps의 네트워크(WAN)로 연결되어 있으며, Venus는 K*Grid Gateway와 100Mbps의 LAN으로 연결되어 있다.

표 2 실험에 사용한 K*Grid의 KISTI Venus, 건국대 Newcluster 시스템

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

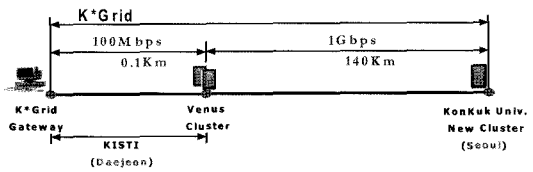

그림 12 동적 재구성 실험에 사용한 K*Grid의 Venus, Newcluster 시스템

실험에 사용한 연산 작업은 143×33×65개의 격자 크기를 갖는 Job 1과 161×41×100개의 격자를 갖는 Job 2로 연산에 필요로 하는 작업파일은 표 3과 같이 구성

표 3 실험에 사용한 작업 파일

작업	파일 설명	파일 구분	파일 크기	작업 총크기
Job 1 (143×33×65개의 격자)	실행코드 파일	cactus_m9	2265Kbytes	15.965Mbytes
	격자좌표 파일	.grd	11066.668Kbytes	
	파라미터 파일	.par	1.411Kbytes	
	RSL 파일	.rsl	0.420Kbytes	
	체크포인팅 파일	.asc×7	376Kbytes×7	
Job 2 (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	

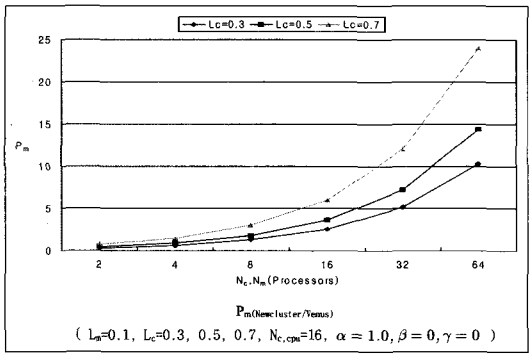

그림 13 식 (1)의 성능지수 값에 따른 $N_{c,cpu}$=16일 때, Venus의 L_c에 대한 Newcluster의 P_m값

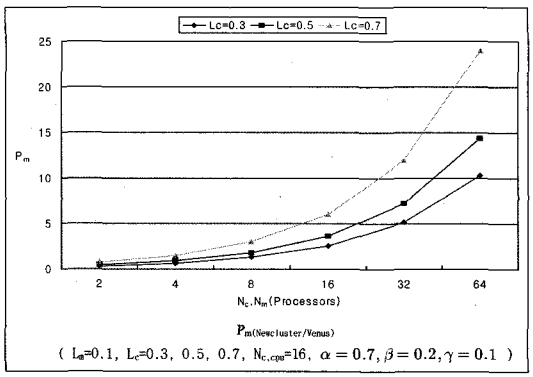

그림 14 식 (1)의 성능지수 값에 따른 $N_{c,cpu}$=16, 가중치가 $\alpha = 0.7, \beta = 0.2, \gamma = 0.1$일 때 Venus의 L_c에 대한 Newcluster의 P_m값

되었다. Job 1의 작업 파일의 크기는 15.97Mbytes이고 Job 2의 작업 파일의 크기는 89.73Mbytes의 크기를 갖는다. Cactus 프레임웍에서 3차원 Euler 방정식을 사용하여 CFD 문제를 10,000번 계산하는 것으로 작업파일은 실행코드 파일, 격자좌표 파일, 파라미터 파일, RSL 파일, 체크 포인팅 파일로 구성되어 있다.

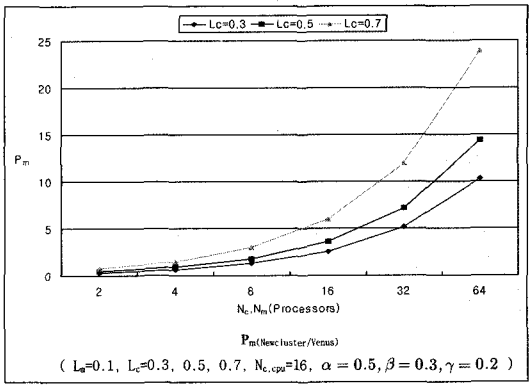

그림 15 식 (1)의 성능지수 값에 따른 $N_{c,cpu}$=16, 가중치가 $\alpha = 0.5, \beta = 0.3, \gamma = 0.2$일 때 Venus의 L_c에 대한 Newcluster의 P_m값

그림 13은 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 Newcluster로 동적재구성을 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 동적재구성을 수행하기 위한 P_m의 값을 갖는 것을 알 수 있다. 그림 14는 가중치가 $\alpha = 0.7, \beta = 0.2, \gamma = 0.1$일 때, 그림 15는 가중치가 $\alpha = 0.7, \beta = 0.2, \gamma = 0.1$ 일 때 P_m의 값을 나타낸 것이다. 가중치의 변화에 따라 P_m값의 차이는 보이고 있지만, 전체적인 경향은 변화가 없음을 알 수 있다.

표 4와 표 5는 다른 사이트의 클러스터에서 새롭게 유휴상태로 발견된 유휴 프로세서를 포함하도록 동적재구성 방법에 대해 실험 한 것이다. 표 4와 표 5에서 **L**은 연산에 참여한 프로세서의 평균 부하를, $\mathbf{N_{cpu}}$는 연산에 참여하는 프로세서의 개수를 의미한다. 표 4에서

표 4 작업 1의 다른 사이트의 유휴 프로세서를 포함하는 동적 재구성

Reconfiguration Sites	L	N_{cpu}	P_m $\alpha=0.7,\beta=0.2,\gamma=0.1$ $(\alpha=0.5,\beta=0.3,\gamma=0.2)$	연산 소요 시간
Venus(4)	L_c=0.025	$N_{c,cpu}$=4	0.55(0.41)	1123분
Venus(8)	L_c=0.17	$N_{c,cpu}$=8	1.68(1.67)	715분
Venus(4)+Newcluster(4)	$L_{m,venus}$=0.005, $L_{m,newcluster}$=0.01	$N_{m,cpu}$=8	21.06(20.39)	704분
Venus(8)+Newcluster(8)	$L_{m,venus}$=0.002, $L_{m,newcluster}$=0.001	$N_{m,cpu}$=16	42.08(41.02)	688분

표 5 작업 2의 다른 사이트의 유휴 프로세서를 포함하는 동적 재구성

Reconfiguration Sites	L	N_{cpu}	P_m $\alpha=0.7,\beta=0.2,\gamma=0.1$ $(\alpha=0.5,\beta=0.3,\gamma=0.2)$	연산 소요 시간
Venus(4)	L_c=0.014	$N_{c,cpu}$=4	0.57(0.415)	2384분59초
Venus(8)	L_c=0.009	$N_{c,cpu}$=8	1.98(1.96)	1429분37초
Venus(4)+Newcluster(4)	$L_{m,venus}$=0.01, $L_{m,newcluster}$=0.001	$N_{m,cpu}$=8	20.81(20.34)	1411분10초
Venus(8)+Newcluster(8)	$L_{m,venus}$=0.1, $L_{m,newcluster}$=0.001	$N_{m,cpu}$=16	41.40(39.53)	1395분53초

Reconfiguration Sites 항목의 괄호 안의 숫자는 연산에 참여한 해당 사이트의 유휴 프로세서의 개수를 의미한다. P_m 항목의 괄호 안의 숫자는 $\alpha=0.5,\beta=0.3,\gamma=0.2$ 일 때의 값으로서 전체적인 경향에는 큰 차이가 없음을 알 수 있다.

첫 번째 실험은 Venus 클러스터에서 유휴 프로세서 4개를 사용하여 연산에 걸린 시간을 측정 하였다. 두 번째 실험은 Venus 클러스터에서 유휴 프로세서 8개를 사용하여 연산에 걸린 시간을 측정 하였다. 세 번째로 Venus의 유휴 프로세서 4개와 Newcluster의 유휴 프로세서 4개를 사용하였다. 네 번째는 Venus의 유휴 프로세서 8개와 Newcluster 8개, 총 16개의 프로세서를 사용한 실험 결과이다. 표 4에서 단축된 연산시간이 비교적 작게 나타난 것은 Venus(8)과 부하의 차가 적은 경우이기 때문이다. 부하의 차가 큰 경우는 연산 시간의 차는 더욱 커지게 된다. 표 5는 작업 2의 동적재구성을 보여주고 있다. Venus(4)는 4개의 프로세서만을 사용한 경우, Venus(8)은 4개의 프로세서를 추가 할당한 경우이다. 보다 우수한 유휴 프로세서를 갖고 있는 사이트를 포함하도록 재구성함으로써 실행시간이 38.7%, 41.4% 단축됨을 알 수 있다.

6. 결론 및 향후 연구

본 논문에서는 K*Grid 환경에서 동적 재구성을 지원하기 위해 응용프로그램 수준의 체크포인팅을 사용하여 연산 작업의 실행 중에 새롭게 발견된 유휴자원을 현재 연산 작업에 동적으로 포함할 수 있도록 지원하는 방법에 대해 연구 하였다. 제안한 방법의 장점은 기존의 동적 재구성과 달리 특정 하드웨어와 운영체제의 지원 없이 글로버스와 Cactus가 실행되는 플랫폼에서 사용가능하다. 또한 응용 프로그램의 수정 없이 재구성을 지원함으로서 유휴 사이트의 연산자원을 효율적으로 활용할 수 있다는 것이다. 본 기법은 그리드 컴퓨팅과 같이 동적으로 연산자원이 변화하는 환경에서 보다 효율적으로 연산자원을 사용할 수 있도록 하고, 현재 연산 작업의 실행 시간을 단축함을 보였다. 차후, 동적 재구성을 통하여 과부하를 갖는 노드를 제거하는 동적 축소와 국내뿐만 아니라 국외 사이트를 포함하여 보다 대규모의 연산 자원을 사용하는 동적인 그리드 컴퓨팅 환경에서 추가의 연구가 수행될 필요가 있다.

참 고 문 헌

[1] Luís Fabrício Wanderley Góes, Carlos Augusto Paiva da Silva Martins, "Reconfigurable Gang Scheduling Algorithm," 10th workshop on Job Scheduling Strategies for parallel processing in conjunction with SIGMETRICS 2004, columbia university, NewYork, NY June 13, 2004.

[2] J. E. Moreira, V. K. Naik, "Dynamic resource management on distributed systems using reconfigurable applications," IBM J. RES. DEVELOP. VOL. 41 NO. 3 MAY 1997

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

[4] Grabrielle 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), Pages 179-188, 2001.
[5] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of Field Programmable Gate Arrays," Proceedings of the IEEE, July 1993.
[6] Kiran Bondalapati, Viktor K. Prasanna, "Reconfigurable Computing Systems," Proceedings of the IEEE, 90(7):1201-17, July 2002.
[7] Kiran Bondalapati and Viktor K. Prasanna, "Reconfigurable Computing: Architectures, Models and Algorithms," CURRENT SCIENCE: Special Section on Computational Science, Vol. 78, no. 7, pp. 828-837, April. 2000.
[8] K. Whisnant, Z. T. Kalbarczyk, R. K. Iyer, "A system model for dynamically reconfigurable software," IBM SYSTEMS JOURNAL, VOL 42, NO 1, 2003.
[9] Compton, K., Hauck, S. "Reconfigurable Computing: A survey of Systems and Software," ACM Computing Survey.(2002)
[10] Dehon, A. "The Density Advantage of Configurable Computing," IEEE Computer, Vol. 33, No.4.(2000).
[11] Góes, L. F. W., Martins, C. A. P. S. RJSSim, "A Reconfigurable Job Scheduling Simulator for Parallel Processing Learning," 33rd ASEE/IEEE Frontiers in Education Conference. Colorado(2003).
[12] J. Jann, L. M. Browning, R. S. Burugula, "Dynamic reconfiguration: Basic building blocks for autonomic computing on IBM pSeries servers," IBM SYSTEMS JOURNAL, VOL 42. NO 1, 2003.
[13] 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.
[14] 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.
[15] Sathish S. Vadhiyar and Jack J. Dongarra, "Self Adaptivity in Grid Computing," Concurrency and Computation: PRACTICE AND EXPERIENCE 2004; 00:1-26.
[16] 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.
[17] 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.
[18] Karl Czajkowski, Ian Foster, Carl Kesselman, Stuart Martin, Warren Smith, Steven Tuecke, "A Resource Management Architecture for Metacomputing Systems," Proc. 4th Workshop on Job Scheduling Strategies for Parallel Processing, pp. 62-82, Springer-Velag, 1998.

김 영 균
정보과학회논문지 : 컴퓨팅의 실제
제 11 권 제 4 호 참조

오 길 호
정보과학회논문지 : 컴퓨팅의 실제
제 11 권 제 4 호 참조

조 금 원
정보과학회논문지 : 컴퓨팅의 실제
제 11 권 제 4 호 참조

나 정 수
2001년 배재대학교 응용수학과 학사
2004년 충남대학교 항공우주공학과 석사
2003년~현재 한국과학기술정보연구원 (KISTI) 위촉 연구원. 관심분야는 항공우주 CFD, 수치해석, PSE 구축