계산 그리드를 위한 커스터마이즈 가능한 글로벌 작업 스케줄러
(Customizable Global Job Scheduler for Computational Grid)

황 선 태 [†] 허 대 영 [††]
(Suntae Hwang) (Dae-young Heo)

요 약 계산 그리드는 다양한 컴퓨팅 자원을 통합한 환경을 제공하며, 그리드 환경은 기존의 컴퓨팅 환경에 비해 매우 복잡하며 다양하다. 그리고 그리드 자원들은 각각 같지 않은 플랫폼과 서로 다른 소프트웨어들을 설치하고 있다. 계산 그리드를 보다 효율적으로 사용하기 위해서는 그리드 자원들을 효과적으로 다룰 수 있는 통합이 필요하다. 본 논문에서는 그리드의 자원을 메타 수준에서 통합하면서 동시에 다양한 정책을 반영할 수 있는 글로벌 스케줄러를 소개한다. 이 글로벌 스케줄러는 기계적인 부분과 세개의 정책으로 구성되어 있다. 기계적인 부분은 적절한 사용자 작업과 계산 자원을 선택하기 위해서 주로 사용자 대기열과 자원 대기열을 검색한다. 이 기계적 부분을 위한 최적화된 알고리즘이 정의되었다. 또한 세개의 정책은 사용자 선택 정책, 자원 선택 정책, 자원 할당 정책으로서 이들은 계산 그리드의 운영을 잠시 중단하고 새로 정의해서 교체 할 수 있다. 예를 들면 사용자 선택 정책은 특정 사용자가 다른 사용자보다 높은 우선 순위를 가지게 하거나 할 수 있고, 자원 선택 정책은 사용자가 요구하는 컴퓨팅 자원에 부합하는 자원을 선택하도록 하며, 자원 할당 정책은 그리드 기반의 통신에서 올 수 있는 부하를 제어하여 극복할 수 있다. 마지막으로, 사용자 선택 정책을 위한 여러 가지 알고리즘을 사용자 형평성만을 고려하여 정의하고 이들의 성능을 측정하여 비교하였다.

키워드 : 그리드, 계산 그리드, 스케줄러, 글로벌 작업 스케줄러, 메타 스케줄러

Abstract Computational grid provides the environment which integrates various computing resources. Grid environment is more complex and various than traditional computing environment, and consists of various resources where various software packages are installed in different platforms. For more efficient usage of computational grid, therefore, some kind of integration is required to manage grid resources more effectively. In this paper, a global scheduler is suggested, which integrates grid resources at meta level with applying various scheduling policies. The global scheduler consists of a mechanical part and three policies. The mechanical part mainly search user queues and resource queues to select appropriate job and computing resource. An algorithm for the mechanical part is defined and optimized. Three policies are user selecting policy, resource selecting policy, and executing policy. These can be defined newly and replaced with new one freely while operation of computational grid is temporarily holding. User selecting policy, for example, can be defined to select a certain user with higher priority than other users, resource selecting policy is for selecting the computing resource which is matched well with user's requirements, and executing policy is to overcome communication overheads on grid middleware. Finally, various algorithms for user selecting policy are defined only in terms of user fairness, and their performances are compared.

Key words : Grid, Computational grid, Scheduler, Global job scheduler, Meta scheduler

· 본 논문은 2006년도 국민대학교 교내연구비를 지원받아 수행된 연구입니다.
† 종신회원 : 국민대학교 컴퓨터학부 교수
sthwang@kookmin.ac.kr
†† 학생회원 : 국민대학교 전산과학
dyheo@cs.kookmin.ac.kr
논문접수 : 2005년 9월 3일
심사완료 : 2006년 4월 7일

1. 서 론

과학/공학 사용자의 컴퓨터를 활용한 연구가 급증하고, 컴퓨터를 활용하여 해결하고자 하는 문제의 크기가 매우 커져가고 있다. 문제를 해결하기 위해서 사용자는 보다 높은 성능과 많은 수의 컴퓨터 자원을 사용하고자 한다.

계산 그리드는 이러한 사용자의 요구사항을 만족 시켜줄 수 있는 환경중의 하나이다. 계산 그리드는 매우 다양한 컴퓨터 자원과 많은 수의 자원을 지원한다. 사용자가 원하는 성능을 가진 컴퓨터 자원이나 원하는 수량만큼의 컴퓨터 자원의 활용이 가능해지고 있는 것이다.

또한 과학/공학 사용자는 오랫동안 검증되고 사용되어온 소프트웨어를 선호한다. 선호되는 소프트웨어는 그 분야에서 문제를 해결하는 데 있어 검증을 받은 소프트웨어들이다. 사용자들은 검증된 소프트웨어를 계산 그리드에서도 사용하기를 원한다.

그러나 계산 그리드는 기존의 컴퓨팅 자원 활용 기술보다 시스템이 복잡해지고 더욱 다양해졌다. 그리드 자원들은 각각 서로 다른 플랫폼을 사용하여 운영되며, 서로 다른 종류의 소프트웨어가 설치되어 있다. 계산 그리드에서 자원들의 이질성은 사용하기 어렵게 만든다. 그러므로 계산 그리드는 자원의 이질성을 극복하고 효율적으로 자원을 활용할 수 있는 메타 수준의 통합이 필요하다.

통합을 위해서는 자원을 기술하는 방법이 단일화 돼야 한다. 자원 기술 정보에는 기본적으로 자원의 플랫폼, 자원의 성능, 자원의 컴퓨팅 작업의 수용 량이 필요하다. 추가적으로 자원에서 수행할 수 있게 설치되어 있는 소프트웨어의 정보가 필요하다.

본 논문에서는 위에서 설명한 자원 기술 정보를 바탕으로 하는 메타 정보 기반의 글로벌 스케줄러를 설계하였다. 제안한 글로벌 스케줄러는 사용자가 선호하는 소프트웨어의 설치 분포와 분포의 변동사항에 대해 적응할 수 있는 스케줄러를 설계하였다. 또한 형평성 및 접근제어를 할 수 있는 사용자에 대한 정책과 보다 빠른 작업 처리를 위한 자원을 선택하는 정책, 통신 부하를 줄이기 위한 자원 할당 정책을 임의로 수립할 수 있도록 고안되었다.

2. 글로벌 스케줄러의 설계

2.1 레거시 소프트웨어 기반의 환경

그리드를 구성하는 각각의 자원은 소프트웨어 라이선스 혹은 자원을 보유한 기관의 내부 정책에 따라서 지원하는 레거시 소프트웨어는 그림 1과 같이 서로 다르다. 즉, 각 자원은 하나의 레거시 소프트웨어를 지원할 수도 있고, 두 개 이상의 레거시 소프트웨어를 지원할 수도 있다.

그리드의 소프트웨어 환경은 동적이다. 그림 2에서 표현되는 영역은 언제든지 변할 수 있다. 자원의 정책에 따라 레거시 소프트웨어를 추가적으로 설치 및 설정할 수 있다.

같은 소프트웨어 환경을 가진 자원 간에도 이질성이

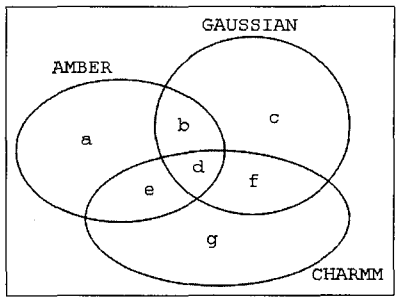

그림 1 레거시 소프트웨어의 환경

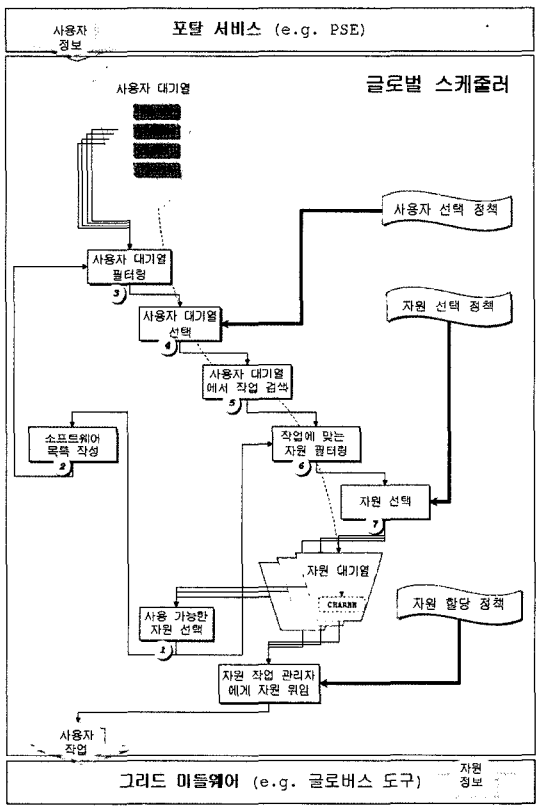

그림 2 글로벌 스케줄러의 구조

존재한다. 각 자원에서 운영되는 플랫폼은 동일하지 않다. 각 자원에서 사용 중인 기계의 특성에 따라, 혹은 자원 제공자의 선호도에 따라 플랫폼은 달라질 수 있다. 플랫폼의 차이는 소프트웨어를 실행하는 방법에 차이를 가져온다. 예를 들어, 소프트웨어를 실행하기 위해 필요한 환경변수를 설정하는 것이 다를 수 있다. 또한 자원 제공자에 따라 설치할 때에, 소프트웨어의 경로가 다르게 된다.

위와 같은 특성은 사용자가 그리드 자원에서 레거시 소프트웨어를 사용하기 어렵게 만든다. 사용자가 직접

그리드의 자원을 활용하기 위해서는 자주 자원의 정보를 수집해야 한다. 이러한 정보의 수집은 레거시 소프트웨어를 사용하는 과학/공학 사용자의 경우, 응용문제를 해결하는 데 부담을 가중시킨다.

2.2 사용자 선택

사용자는 최대한 빨리 응용문제를 해결하기 위해서 보다 성능이 좋은, 보다 많은 컴퓨팅 자원을 요구한다. 대부분의 경우 사용할 수 있는 자원은 사용자의 요구보다 적다. 사용자는 다른 사용자가 자원을 사용하지 않을 때까지 기다려야 하는 불편함을 겪어야 한다.

따라서 스케줄러는 사용자가 요구하는 자원이 현재 준비되어 있지 않더라도, 사용자의 작업을 수용할 수 있는 큐잉 시스템이 필요하다. 그러나 큐잉 시스템만으로는 사용자가 요구하는 조건에 알맞은 자원을 찾아주는 것과 같은 일을 할 수 없다. 스케줄러는 큐잉 시스템에 들어온 사용자의 작업 요청과 가용한 자원들 사이에서 어떤 사용자를 선택하여 자원을 할당할 것인지 결정할 수 있어야 한다.

2.3 자원 활용

그리드의 자원은 매우 다양한 특성과 성격을 가지고 있다. 이러한 자원을 사용자가 모두 파악하는 것은 상당히 어렵다. 또한 각 자원을 이용하는 방법은 자원의 설정에 따라 다르다. 이러한 것은 사용자가 해결하려고 하는 문제의 범위밖에 있는 것이다. 이 문제는 사용자가 그리드의 자원을 활용하는 데, 방해하는 요소가 될 수 있다. 스케줄러는 보다 쉽게 자원을 활용할 수 있도록 지원해야 한다.

2.4 그 외의 오버헤드

그리드 자원과 글로벌 스케줄러는 WAN환경에서 만난다. 글로벌 스케줄러에 의해 작업이 그리드 자원으로 위임될 때, 스케줄러와 그리드 자원은 WAN환경에서 통신이 이루어진다. 스케줄러는 그리드 자원과 통신하기 위해서는 표준적으로 사용하는 글로버스 도구[1-3] 같은 그리드 미들웨어를 사용해야 한다. 미들웨어를 사용한 통신은 각각의 접속마다 사용자의 인증 과정과 해당 사용자를 자원의 계정으로 연결하는 작업이 포함되어 있다. 일반적으로 WAN환경에서 통신은 빠르지 않은 데, 이 환경에 부수적인 통신이 추가적으로 존재하기 때문에 스케줄러가 짧은 시간에 많은 작업을 위임하려고 한다면 부하가 발생할 수 있다.

3. 글로벌 스케줄러의 구현

3.1 사용자 정보와 자원 정보

레거시 소프트웨어를 지원하기 위해서, 글로벌 스케줄러는 그리드의 각 자원에서 무슨 레거시 소프트웨어를 사용할 수 있는지, 어떻게 실행할 수 있는지 알아야 한다. 글로버스 미들웨어에 있는 정보 서비스를 이용한다. 정보 서비스는 자원의 기본적인 정보, 자원의 성능을 제공한다. 본 논문에서는 각 자원의 소프트웨어 환경을 정보 서비스에 등록하여 스케줄러에서 사용할 수 있도록 한다. 소프트웨어는 종류와 버전으로 구분한다. 또한 필요에 따라 소프트웨어의 설치 옵션에 대한 정보도 기술한다. 각 자원에는 소프트웨어의 실행 경로까지 제공하여야만, 글로벌 스케줄러는 소프트웨어를 실행할 수 있다.

3.2 구성 요소

사용자 대기열 : 글로벌 스케줄러에서 그리드 사용자가 가지는 대기열(Queue)이다. 각 사용자는 독립적으로 하나를 가진다. 대기열에서 작업은 들어온 순서대로 정렬된다. 이것은 사용자에 의해 순서가 다시 정렬되어 질 수 있다. 대기열에서 작업이 나가는 것은 정렬된 순서로 검색하여 가장 먼저 수행 가능한 것부터 나간다.

자원 대기열 : 그리드의 각 자원을 대표하는 대기열(Spool)이다. 자원 대기열의 목적은 글로벌 스케줄러와 각 자원과의 통신 과정에 발생 하는 오버헤드를 스케줄 시점에서 고려하지 않기 위해 존재한다. 자원 대기열은 컴퓨터 자원의 구성 형태(SMP 혹은 MPP 등)를 고려하지 않는다. 단지 보유하고 있는 CPU의 수만을 고려하며 보유한 각 CPU를 슬롯형태로 표현한다. 각 슬롯에는 한 개 혹은 둘 이상의 작업이 할당되어 질 수 있다. 자원 대기열에서 모든 CPU 슬롯이 사용 중일 때에, 자원 대기열은 가용불가하다고 말하며, 단 한 개의 슬롯이라도 비어 있다면, 가용하다고 이야기한다.

사용자 선택 정책 : 스케줄러에 요청된 사용자의 작업들 사이에서 어떠한 사용자의 작업을 우선적으로 처리할 것인가를 결정해야 한다. 사용자 선택 정책은 우선 처리해야 하는 사용자의 작업을 선별하기 위해 제공한다. 정책은 특정 사용자의 자원 사용을 제한하거나, 특정 사용자의 자원 사용률을 높이기 위해 높은 우선순위를 부여할 수도 있으며, 형평성을 중시할 수도 있다.

자원 선택 정책 : 선택된 작업이 어떠한 자원에서 수행하게 할 것인지에 대한 정책을 제공한다. 예를 들어서 현재 사용 가능한 자원 중, 가장 성능이 좋은 자원을 선택하거나, 가장 먼저 알맞은 자원을 선택하는 것을 결정한다.

자원 할당 정책 : 자원과 통신할 때에, 얼마나 많은 작업을 한꺼번에 보낼 것인가에 대한 정책을 제공한다. 예를 들어, 통신에 대한 오버헤드를 고려하여, 수행시간이 짧은 작업은 한 번에 많이 보내는 것을 결정한다.

3.3 스케줄

표 1의 알고리즘은 그림 2를 반영하여 표현한 것이다. 다음은 스케줄이 실행되기 전에 스케줄러는 사용 가능한 자원이 존재한다는 것을 알고 있다고 가정한다.

표 1 스케줄 알고리즘

User_Policy() : 사용자 정책
Resource_Fit_Policy() : 자원 선택 정책
Resource_allocate_Policy() : 자원 할당 정책

Schedule()
U : 사용자 대기열 집합
u : u ∈ U, 사용자 대기열
R : 자원 대기열 집합
S : 소프트웨어 집합
r : r ∈ R, 자원 대기열
j : j ∈ u, 작업
BEGIN
① R_{avail} = R에서 가용한 자원 수집
② S_{avail} = R_{avail}의 소프트웨어 수집
U_{avail} = U
REPEAT
③ u = **User_Policy**(U_{avail})
U_{avail} = U_{avail} - u
④ j = **SearchBySoftware**(u, S_{avail})
UNTIL j≠ϕ
R_{set} = ϕ
s_j= **getSoftware**(j)
⑤ FOR r in R_{avail}
s_r= **getSoftware**(r)
IF **isExcutable**(s_j,s_r) THEN
R_{set} = R_{set}∪ {r}
END IF
END FOR
⑥ R_{fit} = **Resource_Fit_Policy**(R_{set})
enque j to R_{fit}
END

① 사용 가능한 자원이 있는지 확인 한다.
② 수집한 자원 대기열의 집합에서 소프트웨어 목록을 작성한다. (가용 소프트웨어 비트 생성)
③ 사용자 선택 정책을 적용하여 사용자 대기열을 선택한다.
④ 선택된 사용자 대기열에서 소프트웨어 목록에 있는 소프트웨어를 사용하는 작업을 검색한다. 만약 수행 가능한 작업이 없다며 ③부터 반복한다.
⑤ 선택된 작업(j)의 소프트웨어를 수행할 수 있는 자원의 대기열 집합(R_{set})을 구한다.
⑥ Rf에서 자원 선택 정책을 적용하여 가장 알맞은 자원 대기열(R_{fit})를 선택한다.
⑦ R_{fit}에 j를 넣는다.

3.3.1 가용 소프트웨어의 수집

스케줄러는 그림 3에서처럼 모든 소프트웨어에 대한 비트 값을 가지고 있다. 각 자원의 대기열에는 수행 가능한 소프트웨어의 비트의 합을 가지고 있다. 따라서 현재 가용한 CPU 슬롯을 가지고 있는 자원 대기열들의 소프트웨어 비트를 모두 합한 것이 현재 수행이 가능한 소프트웨어 목록이 된다.

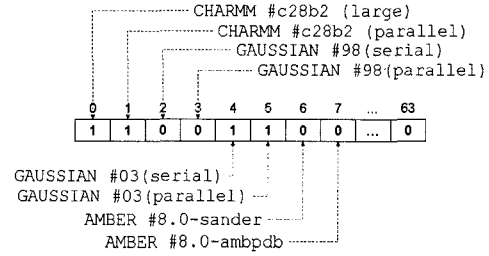

그림 3 소프트웨어의 비트 표현

3.3.2 사용자 대기열 필터링

사용자 대기열 필터링의 목적은 가용한 CPU 슬롯을 가지고 있는 자원에서 수행할 수 있는 작업을 가지고 있는 사용자 대기열을 찾기 위한 것이다. 어떤 사용자 대기열에 전 단계에서 수집한 가용 소프트웨어 중에 있는 소프트웨어를 사용하는 작업이 단 한 개도 없다면, 사용자 대기열은 정책 적용에서 제외되어야 한다.

사용자 형평성의 정책 적용 비용이 클 경우에는 정책을 적용하기 전에 사용자 대기열에 가용 소프트웨어를 사용하는 작업이 한 개라도 있다는 것을 보장해야 한다. 그러나 정책 적용 비용이 무시할 수 있거나 크지 않을 경우에는 대기열 필터링 비용이 오히려 클 수도 있다. 전체 사용자 대기열에서 정책을 적용한 후 선택된 사용자 대기열에 적용할 수 있는 작업이 없을 경우, 현재 선택된 사용자 대기열을 소거한 후 정책을 다시 적용할 수 있다.

3.3.3 선택된 사용자 대기열에서 작업 검색

선택된 사용자 대기열에서 작업을 검색하는 것은 가용 소프트웨어 비트와 작업이 요구한 소프트웨어 비트를 비교하여 선택할 수 있다.

사용자 대기열 필터링을 수행하지 않은 경우, 선택된 사용자 대기열에서 작업 검색이 실패할 가능성이 있다. 작업 검색을 수행하지 않고 사용자 대기열에 원하는 작업이 있는 지 확인할 수 있다면, 검색 실패에 대한 비용을 줄일 수 있다.

대부분의 사용자는 과거에 사용했던 소프트웨어를 다시 사용할 가능성이 매우 높다. 대기열에서 과거 사용했던 소프트웨어를 사용하는 작업이 없을 수는 있지만, 확률적으로 있을 가능성이 매우 높다고 말할 수 있다. 작업이 사용자 대기열에 추가될 때마다 사용자 대기열에 사용자가 사용하는 소프트웨어 비트를 기록하고, 대기열에서 작업 검색이 이루어지기 전에 현재 가용 소프트웨어 비트와 대기열이 과거에 사용하였던 소프트웨어 비트를 비교하여 대기열에 수행할 수 있는 작업이 있는지 검사할 수 있다.

3.3.4 전체 알고리즘의 비용

전체 알고리즘의 비용은 그림 2의 1~7단계의 비용을 합한 것이다. 사용자 형평성 정책의 비용은 $P(User)$로 정의하고, 자원 선택 정책의 비용은 $P(Resource)$로 정의한다. $N(x)$을 x의 수량으로 정의한다. 식 (1)은 각 단계에서 소비하는 비용을 합한 전체 알고리즘 비용이다.

$$O(N(R_{avail})) + O(N(User)\times(P(User)+N(Job_{User}))) + N(R_{avail}) + O(P(Resource)) \quad (1)$$

제1단계, 사용 가능한 자원 선택 : 실시간으로 검사하여 항상 가지고 있을 수 있다. 따라서 비용이 들지 않는다.

제2단계, 소프트웨어 목록의 작성 : 가용 자원을 R_{avail}로 하면, 각 자원의 소프트웨어 비트를 더하는 것으로 비용은 $O(N(R_{avail}))$이다.

제3,4단계, 사용자 대기열 필터링과 정책의 적용 : 최악의 경우 전체 사용자 대기열 수만큼 정책을 적용해야 하기 때문에 $O(N(User)\times P(User))$이 든다.

제5단계, 작업 검색 : 사용자 대기열에 있는 작업의 수를 $N(Job_{User})$라고 하면, 가용 소프트웨어 비트와 작업의 비트를 최악의 경우 $N(Job_{User})$번 비교해야 함으로 비용은 $O(N(Job_{User}))$이다.

제6,7단계, 작업에 맞는 자원 수집 및 자원 선택 : 가용한 자원(R_{avail})중에서 선택된 작업(X_{Job})의 소프트웨어를 실행할 수 있는 모든 자원을 찾는 것의 비용은 $N(R_{avail})$이다. 그리고 자원 선택은 자원 선택 정책 은 $O(P(Resource))$ 비용이 든다.

3.4 구현

본 논문에서 제안한 스케줄러는 그림 4에서처럼 대기열 관리자와 정책 관리자, 그리고 스케줄러와 수행자 모듈로 나눈다. 사용자 대기열과 자원 대기열은 각각의 관리자가 관리하게 된다. 대기열 관리자는 그림 5에서 보여주는 것과 같이 관리하는 대기열에서 일어나는 이벤트를 발견하고 처리할 수 있다. 자원 대기열 관리자는 어떤 자원이 가용한 상태인 지 관리하여 스케줄을 수행할 때에 가용 자원을 수집할 때 드는 비용을 제거해준다.

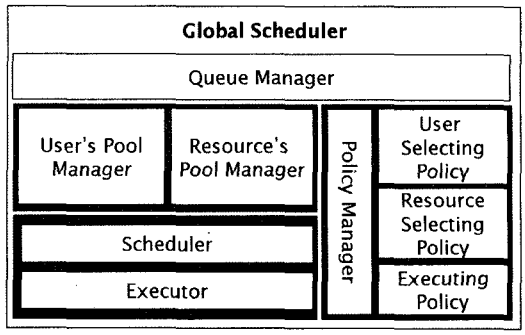

그림 4 글로벌 스케줄러의 구성도

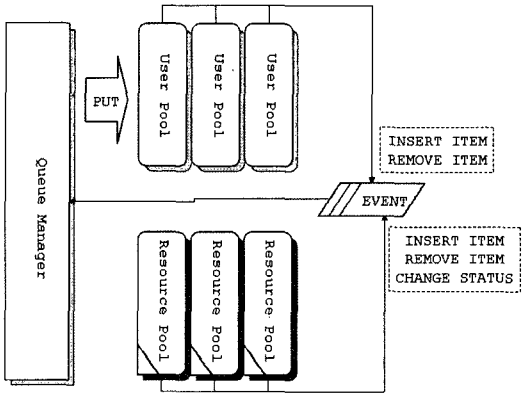

그림 5 대기열의 이벤트 통지

큐 관리자는 스케줄러에 존재하는 사용자 대기열을 추상화하여 준다. 외부시스템에서 작업을 입력하면, 큐 관리자는 입력한 사용자의 사용자 대기열을 검색하여 해당 대기열에 작업을 넣는다. 또한 외부시스템이 사용자 대기열을 제어 및 감시 할 수 있는 인터페이스를 제공한다.

스케줄러 모듈은 정책 관리자로부터 사용자 정책과 자원 선택 정책을 가져와서 적용한다. 3.3절에서 기술한 스케줄을 구현한 것으로 주기적으로 수행된다. 수행자 모듈은 정책 관리자에서 자원 할당 정책을 가져와서 적용한다. 본 논문의 실험을 위해서 가상으로 실행하는 것을 흉내 내었지만, 실제에 적용할 때에는 글로버스 미들웨어와 연동하여 사용하게 된다.

그림 6은 정책을 임의로 정의하여 사용할 수 있도록 하는 인터페이스를 보여준다. 정책 수립은 추상화된 인터페이스를 상속하여 Apply 메서드를 구현하면 된다. 구현된 정책은 정책 관리자 모듈에 의해 초기화 되어, 글로벌 스케줄러에 적용이 된다.

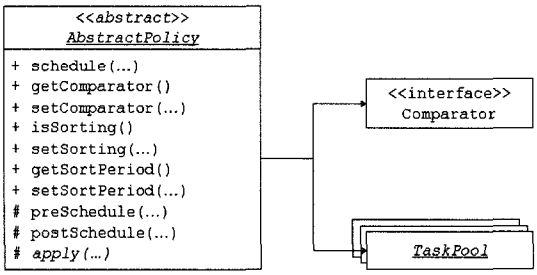

그림 6 추상화 정책 객체

4. 스케줄링 정책

4.1 사용자 선택 정책

본 논문에서 제안한 스케줄러는 사용자가 위임한 작업을 실행할 때에, 사용자에 대한 다양한 정책을 반영할 수 있다. 예로 특정 사용자에 대해서 높은 우선순위를 부여해서 빠르게 작업을 수행할 수 있도록 할 수 있다. 본 논문에서는 사용자의 형평성만을 고려하여 스케줄링 정책을 비교하여 보았다.

4.2 자원 선택 정책

작업이 요구하는 자원의 특성이 존재할 수 있다. 작업을 수행하는 데에, 일정 이상의 보조 저장 공간을 요구하거나, 일정 이상의 주기억 장치의 성능을 요구할 수 있다. 작업이 요구하는 가장 이상적인 자원을 선택할 수 있어야 할 것이다.

4.3 작업 할당 정책

작업의 자원으로 위임은 WAN환경에서 수행된다. WAN환경은 일반적으로 빠르지 않다. 즉, 작업 위임에 걸리는 부하를 계산하여 작업 수행을 결정해야 한다.

$$D_L = \frac{D_C}{J_T} \tag{2}$$

식 (2)에서 D_C는 통신에 걸리는 시간, J_T는 작업 수행 시간이고 D_L는 부하의 크기를 나타낸다. D_C의 경우 대부분 일정한 크기를 가짐으로 J_T의 값에 따라 부하 D_L이 결정된다. J_T의 크기가 클 경우에는 D_L의 값이 무시되어 질 정도로 작다. 하지만 J_T의 크기가 D_C의 크기와 비교해서 큰 차이가 나지 않을 경우, D_L의 크기는 무시할 수 없다. D_L의 기준치보다 클 경우 D_L의 값을 낮출 수 있는 정책을 반영해야 한다.

5. 실험결과

5.1 실험 환경

본 논문이 제안한 스케줄러의 사용자 선택 정책에 모든 사용자가 균등하게 자원을 사용할 수 있는 형평성만을 고려한 알고리즘을 적용하고자 한다. 자원 선택 정책은 무작위 선택으로 하며, 자원 할당 정책은 고려하지 않는다. 형평성만을 고려한 사용자 선택 정책은 다음과 같이 한다.

① 무작위 선택(Random)
② 순회 선택(Round Robin)
③ 최빈값 선택(Least Frequently Used)
④ 균등 우선 최빈값 선택(1/N + LFU)

균등 우선 최빈값 선택 정책은 현재 사용자들이 사용하고 있는 자원의 수를 균등하게 하는 것을 우선으로 하고, 동일할 경우 최빈값 선택 정책을 적용하는 것이다. 본 논문에서는 레거시 소프트웨어 종류를 총 4가지로 하였고, 표 2에서처럼 4가지의 레거시 소프트웨어의 자원 분포 환경으로 실험한다.

표 2 레거시 소프트웨어 환경

	자원의 수	레거시 소프트웨어 분포
실험1	8, 9	동일 분포
실험2	8	배타적 분포(자원 각각 한 가지 소프트웨어 분포)
실험3	8	부분 교차 분포(자원 각각 두 가지 소프트웨어 분포 및 소프트웨어 부분 중첩)
실험4	9	혼합 분포(동일 분포 + 부분 교차 분포)

그림 7과 같이 레거시 소프트웨어 분포 환경은 모두 4가지 형태를 가진다. 첫 번째 동일 분포 환경은 모든 자원이 4가지 소프트웨어를 각 자원이 모두 가지고 있는 환경이다. 배타적 분포 환경은 각 자원이 한 가지 소프트웨어를 가지고 있는 환경이다. 부분 교차 분포 환경은 각 자원이 두 가지 소프트웨어를 가지고 있고, 각 소프트웨어는 두 자원에 설치되어 있는 환경이다. 그리고 한 가지 소프트웨어에 대한 동일 분포 환경과 나머지 세 가지 소프트웨어에 대한 부분 교차 분포 환경이 같이 있는 혼합 분포 환경을 구성하여 시뮬레이션 하였다.

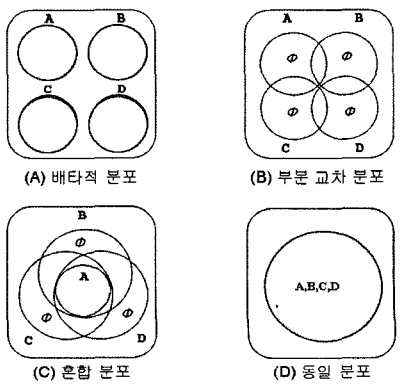

그림 7 소프트웨어의 설치 분포에 따른 자원 구성

대기열이 존재하기 때문에, 사용자가 작업 수행을 한 번에 요청하는 일이 많은 것으로 가정하여, 사용자의 작업 수행 행태는 한 번에 수행하고자 하는 모든 작업을 수행요청 하도록 하였다. 사용자를 두 개의 그룹으로 나누고 각 그룹을 시작 그룹과 후속 그룹으로 한다. 각 그룹에 20명이 작업을 수행하는 것으로 하여 시뮬레이션 하였다. 후속 그룹은 시작 그룹이 작업 수행한 후 3주 후에 시작하도록 하였다. 각 작업은 7일에서 10일 정도 수행된다. 1인당 20개의 작업을 수행한다.

5.2 실험 결과

5.2.1 소프트웨어 동일 분포 환경

그림 8~11은 동일 분포 환경에서 실험하였을 때 결과이다. 그래프의 X축은 사용자가 사용하고 있는 CPU의 수이다. Y축은 자원을 사용한 날 수이다. 실선으로 나타난 데이타는 시작 그룹이 1인당 사용하고 있는 평균 CPU 수이고, 점선으로 나타난 데이타는 후속 그룹이 1인당 사용하고 있는 평균 CPU 수이다.

그림 8-12를 보면 후속그룹은 21일에 작업 수행을 요청한 후 10일정도 경과한 후에 사용하고 있는 CPU수가 증가하는 것을 볼 수 있다. 무작위 선택 정책은 시작 그룹과 후속그룹의 CPU 평균 사용량이 비슷해지는 시점이 50일 정도 되며, 두 그룹이 지속적으로 균등한 사용하지 못하고 있다. 순회 정책의 경우에는 무작위 정책보다는 늦은 60일 정도에서 시작 그룹과 후속 그룹의 CPU 평균 사용량이 비슷해진다. 그러나 무작위 정책과는 달리 두 그룹이 지속적으로 균등하게 사용한다는 것을 알 수 있다. 최빈값 정책은 순회 정책 보다 빠른 50일 정도에서 두 그룹이 평균 CPU 사용이 비슷해지며 순회 정책과 같이 지속적으로 두 그룹이 평균을 유지하는 것을 볼 수 있다. 마지막으로 균등 우선 최빈값 정책은 앞선 모든 정책 보다 빠른 40일경에 평균 CPU 사용이 비슷해지며, 보다 안정적으로 두 그룹이 평균을 유지하는 것을 볼 수 있다.

표 3은 각 동일 분포 자원 환경에서 시작 그룹과 후

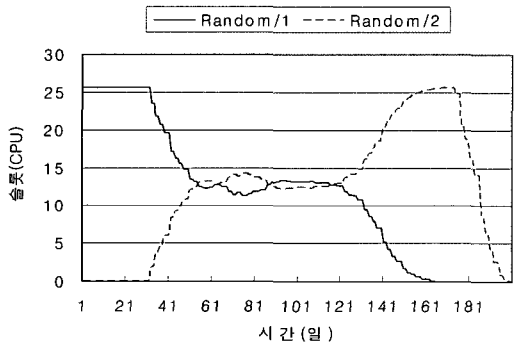

그림 8 무작위 선택 정책. 그룹 점유율 변화

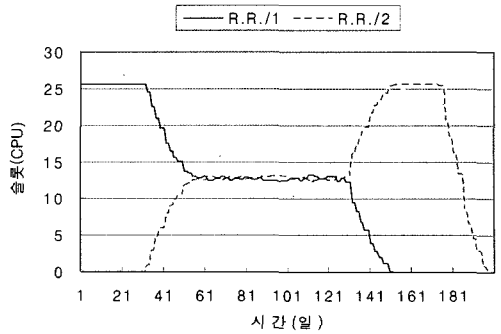

그림 9 순회 정책. 그룹 점유율 변화

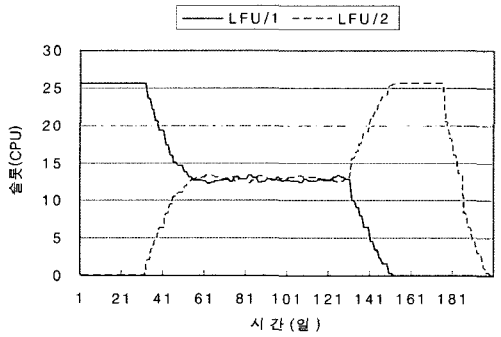

그림 10 최빈값정책. 그룹 점유율 변화

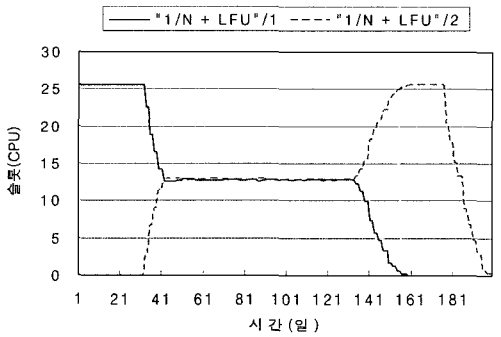

그림 11 균등 우선 최빈값 정책. 그룹 점유율 변화

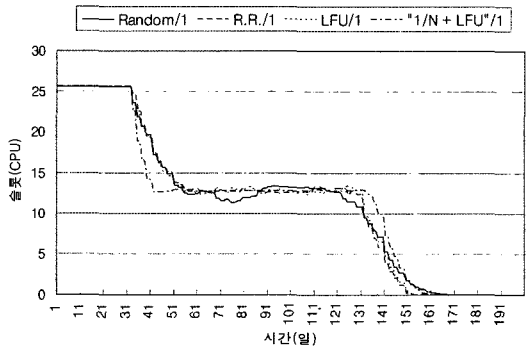

그림 12 정책별 시작 그룹의 평균 CPU 사용 변화 비교

표 3 동일 분포 환경에서 표본 구간 비교

	정책	표준 편차
실험1 (동일)	RANDOM	3.308
	R.R.	1.307
	LFU	1.306
	1/N + LFU	0.500

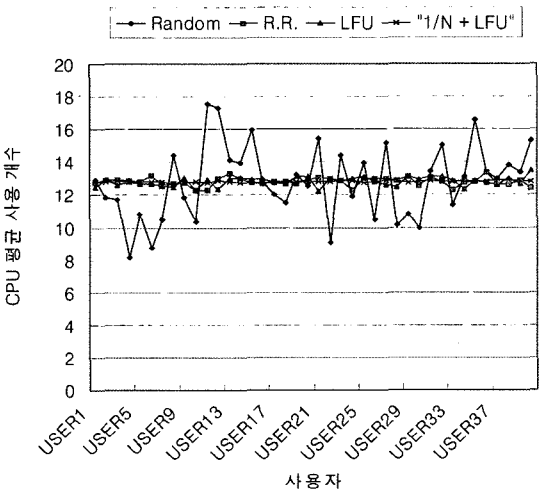

그림 13 동일 분포 자원에서 표본 구간의 사용자별 CPU 평균 사용에 대한 정책 비교

속 그룹의 평균점이 처음으로 같았던 부분에서 마지막으로 같았던 평균점을 표본 구간으로 하여 사용자의 CPU 사용에 대한 표준편차를 구한 것이다. 무작위 선택 정책의 경우 표본 구간에서 표준 편차가 다른 정책에 비해 매우 크지만, 균등 우선 최빈값 정책은 표준 편차가 1 이하로 모든 사용자가 거의 같은 수의 CPU를 사용하게 한다. 그림 13은 표본 구간에서 각 사용자가 평균적으로 사용한 CPU 개수를 나타낸다. 무작위 선택 정책은 공평하지 못하게 CPU 자원이 분배되었고, 균등 우선 최빈값 정책이 가장 고르게 CPU자원이 분배되었다.

5.2.2 다른 형태의 소프트웨어 분포 환경

동일 분포 환경에서는 무작위 선택 정책을 제외한 정책들은 거의 대동소이한 결과를 보여준다. 그러나 그리드 환경에서는 동일 분포 환경이 항상 유지되기 어렵다. 따라서 다른 환경에서도 같은 실험을 수행하였다.

표 4는 배타적 분포, 부분 교차 분포, 혼합 분포 환경에서 실험한 결과를 보여준다. 표 3은 시작 그룹과 후속 그룹이 평균점을 이루었던 구간을 표본 구간으로 하여 개별 사용자의 CPU 사용에 대한 표준 편차를 구한 것이다. 순회 정책은 동일 분포 환경과 가장 큰 차이를 보이고 있다.

그림 14~16은 소프트웨어 분포 환경별 표본 구간에

표 4 각 분포 환경에서 균형 구간의 비교

	정책	표준 편차
실험2 (배타적)	RANDOM	3.519
	R.R.	2.675
	LFU	1.241
	1/N + LFU	0.436
실험3 (부분교차)	RANDOM	3.614
	R.R.	1.868
	LFU	1.336
	1/N + LFU	0.452
실험4 (혼합)	RANDOM	3.840
	R.R.	2.565
	LFU	1.448
	1/N + LFU	0.498

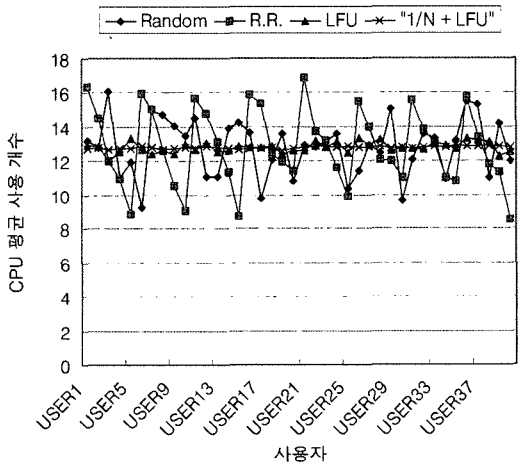

그림 14 배타적 분포(실험 2) 환경에서 표본 구간의 사용자별 CPU 사용 평균에 대한 정책 비교

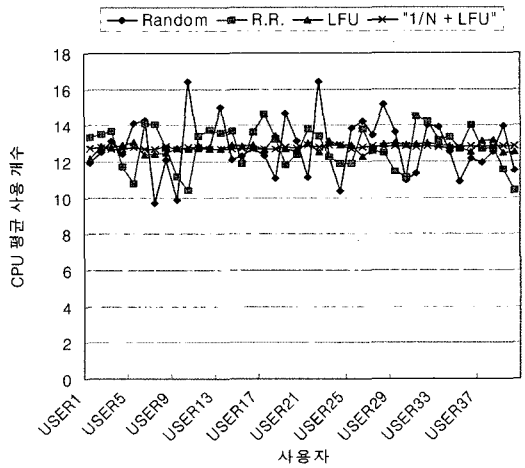

그림 15 부분 교차 분포(실험 3) 환경에서 표본 구간의 사용자별 CPU 사용 평균에 대한 정책 비교

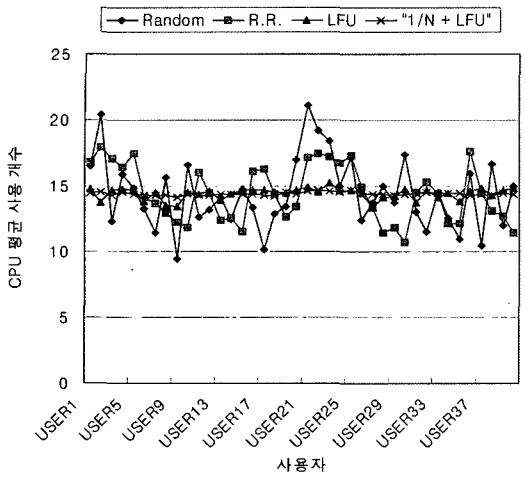

그림 16 동일 분포와 부분 교차 분포의 혼합 분포(실험 4) 환경에서 표본구간의 사용자별 CPU 사용 평균에 대한 정책 비교

서 사용자의 CPU 평균 사용량을 보여준다. 순회 정책은 자원의 분포환경에 따라서 매우 성능의 변화가 심해, 무작위 선택 정책과 비슷하게 나타나기도 한다. 최빈값 정책과 균등 우선 최빈값 정책은 모든 분포 상황에서 적응적이며, 균등 우선 최빈값 정책은 매우 안정화되어 있는 것을 알 수 있다. 특히 배타적 분포 환경에서 순회 정책은 스케줄링 순서가 정해져있기 때문에 앞선 사용자가 항상 먼저 자원을 선점하는 것을 확인할 수 있다.

6. 관련연구

6.1 CSF

CSF(Community Scheduler Framework)[3]은 글로버스 도구에 포함되어 있는 WSRF[4,5]와 호환되는 그리드 수준의 메타 스케줄링 프레임워크이다. CSF는 그리드 사용자를 위해 작업 수행을 요청, 고급화된 자원 예약을 요청할 수 있는 인터페이스와 도구를 제공한다. 뿐만 아니라, 그리드 수준의 서로 다른 스케줄 정책을 반영할 수 있도록 프로그램 인터페이스를 제공한다.

본 논문에서 제안한 스케줄러와 CSF는 매우 유사한 구조를 가지고 있다. CSF는 큐잉 시스템과 그것을 사용할 수 있는 단순한 스케줄 인터페이스를 구현하도록 설계되어 있다. 즉, 스케줄 기능을 구현하면서 정책을 반영해야 한다. 그러나 본 논문에서 제안한 스케줄러는 기계적인 스케줄 기능을 구현되어 있어, 정책만을 구현하여 반영할 수 있도록 설계되어 있다.

6.2 CONDOR-G

Condor-G[6]은 지역 스케줄러인 Condor[6-9]의 기능을 확장한 것으로 Condor의 스케줄링의 알고리즘과 기법들을 그대로 사용하여 그리드 환경 중 글로버스 도구에 맞게 확장된 형태의 작업 스케줄러이다. Condor-G는 Condor에서 사용한 작업 기술 방법을 확장해서 사용하며, Condor에서 확장될 수 있는 부가적인 기술들을 사용할 수 있다. 예를 들어 DAG를 처리할 수 있는 DAGMan을 Condor-G에서도 사용할 수 있게 되어있다.

Condor는 각 단말 노드에서 정보를 수집하는 데몬이 설치되어있어 작업 할당을 수행하기 때문에 작업을 수행하기 위한 최적의 자원을 할당하는 기능을 가지고 있다. 그러나 Condor-G는 Condor와는 다르게 사용자에 의해 어떠한 미들웨어를 사용하고 어떤 자원을 사용할 것인지 미리 기술해야 한다. Condor-G는 Condor 사용자에게 같은 방법으로 그리드 환경을 사용할 수 있는 방법을 제공하는 것이 한계이다.

본 논문에서 제안한 글로벌 스케줄러는 Condor-G와 연동할 수 있다. Condor-G에서는 자원 선택을 할 수 있는 기능이 없기 때문에 본 논문의 자원 선택 정책을 이용하여 보완할 수 있다. 제안한 글로벌 스케줄러는 그리드 자원 정보를 수집한 것을 통하여, 어떤 그리드 자원에서 실행할 것인지 결정하고, 결정된 정보는 Condor-G에게 전송하여 Condor-G에서 작업이 수행되도록 할 수 있다.

6.3 UNiCORE

UNiCORE[10-13]은 그리드를 통합된 인터페이스로 자원에 대한 접근을 허용한다. UNiCORE에서 지원하는 작업 AJO(Abstract Job Object)를 정의하고, 수행할 수 있도록 설계되었다. 작업을 설계할 때에, 자원에서 소프트웨어가 어디에 설치되어 있는지 알지 않게 하기 위해 소프트웨어 이름을 대표 이름으로 기술하게 하고 있다. 이 이름은 실제 작업 수행할 때에 실행 가능한 이름(파일명)으로 치환된다[14,15].

UNiCORE는 자원을 할당하는 정책을 가지고 있지 않다. 스케줄러는 가지고 있지 않지만, 작업에 알맞은 자원을 선별하기 위해 작업이 요구하는 자원의 성능 요인을 설정할 수 있도록 제공하고 있다. 자원에 대한 요구사항은 자원 목록을 표시할 때 반영되고, 사용자는 목록 중 하나의 자원에 보내어 수행할 수 있다.

본 논문에서 UNiCORE의 사용자의 자원 요구사항에 맞는 자원을 검색하여 주는 기능을 자원 선택 정책에 반영하여 일을 수행할 수 있을 것이다.

7. 결론 및 향후 과제

계산 그리드의 다양한 컴퓨팅 자원 환경은 기존의 컴퓨팅 환경에 비해 매우 복잡하고 다양해졌다. 그리드 자원들은 서로 다른 플랫폼과 서로 다른 소프트웨어들을 설치하고 있다. 자원의 이질적인 환경은 자원을 사용하

기 어렵게 하기 때문에, 자원의 이질성을 극복할 수 있는 메타 수준의 통합이 필요하다.

본 논문에서는 자원의 통합을 위해서 메타 수준의 글로벌 스케줄러를 설계하였다. 제안된 글로벌 스케줄러는 통합할 자원의 기술 정보를 그리드 자원을 물리적인 성능과 작업의 수용량, 운용중인 플랫폼뿐만 아니라 사용자가 사용하는 응용문제를 해결할 수 있도록 설치된 소프트웨어 종류까지 확장하였다.

본 논문의 스케줄러는 사용 목적에 따라 정책이 다르게 적용될 수 있도록 지원하고 있다. 사용자 정책, 자원 선택 정책, 자원 할당 정책을 임의로 수립할 수 있다. 사용자 정책을 통하여 형평성이나 접근 제어를 수행할 수 있고, 자원 선택 정책을 통하여 작업 수행에 가장 적합한 자원 검색을 지원할 수 있으며, 자원 할당 정책을 통하여, 서로 다른 미들웨어의 사용이나 미들웨어에서 발생하는 통신 부하에 대하여 고려할 수 있다.

본 논문의 스케줄러를 이용하여 사용자 형평성을 중시하는 정책을 사용하여 실험을 해보았다. 실험에서는 무작위 정책, 순회 정책, 최빈값 정책, 균등 우선 최빈값 정책을 사용하였다. 이 중에서 최빈값 정책과 균등 우선 최빈값 정책이 사용자 형평성면에서 우수한 성능을 나타냈으며 균등 우선 최빈값 정책이 가장 우수하였다.

참 고 문 헌

[1] I. Foster, C. Keselman, S. Tuecke, "The Anatomy of the Grid," International J. Supercomputer Applications. 2001.

[2] I. Foster, C. Kesselman, "The Globus Project: A Status Report," Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop, pp. 4-18, 1998. Describes the status of the Globus system as of early 1998.

[3] Globus Toolkit, CSF, http://www.globus.org/

[4] Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Global Grid Forum, June 22, 2002.

[5] Web Services Resource Framework Specification, http://www.oasis-open.org/committees/wsrf

[6] Condor, Condor-G, http://www.cs.wisc.edu/condor/

[7] M. J. Litzkow and M. Livny, "Experience With The Condor Distributed Batch System," Proceedings of the IEEE Workshop on Experimental Distributed Systems, pp.97-101, October 1990.

[8] M. C. Ferris and T. S. Munson, "Modeling languages and Condor: metacomputing for optimization," Mathematical Programming, 88(3), pp. 487-505, 2000.

[9] M. J. Litzkow and M. Livny, "Experience With The Condor Distributed Batch System," Proceedings of the IEEE Workshop on Experimental Distributed Systems, pp.97-101, October 1990.

[10] Mathide Romberg, "The UNICORE Grid infrastructure," Scientific Programming, 10(2), pp. 149-157, 2002.

[11] A.Reinefeld, H.Stuben, T.Steinke, and W.Baumann, "Models for Specifying Dis-tributed Computer Resources in UNICORE," First European Grid Forum Meeting,Proceedings of the ISThmus 2000 / EUNIS 2000 Conference, Poznan, April 2000.

[12] M.Romberg, "The UNICORE Architecture: Seamless Access to Distributed Re-sources," Proceedings of the Eighth IEEE International Symposium on High Per-formance Distributed Computing, August 1999, pp.287-293.

[13] Dietmar W. Erwin, "UNICORE - a Grid computing environment."

[14] John Brook, Donald Fellows, Kevin Garwood, Carole Goble (2004), "Semantic matching of Grid-resource descriptions," 2nd EUROPEAN ACROSS GRIDS CONFERENCE, Nicosia, Cyprus, Jan. 28-30, 2004.

[15] Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid InformationServices for Resource Sharing," in Proceedings of HPDC-10, IEEE Press, 2001.

[16] M. Hovestadt, O. Kao, A. Keller, and A. Streit, "Scheduling in HPC Resource Management Systems:Queuing vs. Planning," In D. G. Feitelson and L. Rudolph, editors, Proc. of the 9th Workshop on JobScheduling Strategies for Parallel Processing, volume 2862 of Lecture Notes in Computer Science, pages1-20. Springer, 2003.

[17] J. Schopf, "Ten Actions When Grid Scheduling - The User as a Grid Scheduler," In J. Nabrzyski,J. Schopf, and J. Weglarz, editors, Grid Resource Management-State of the Art and Future Trends, pages 15-23. Kluwer Academic Publishers, 2004.

[18] G. Quecke and W. Ziegler, "MeSch - An Approach to Resource Management in a Distributed Environment," In Proc. of 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000), volume 1971of Lecture Notes in Computer Science, pages 47-54. Springer, 2000.

황 선 태

1985년 서울대학교 컴퓨터공학과(학사)
1987년 서울대학교 컴퓨터공학과(석사)
1996년 Manchester University(PhD)
1997년~ 국민대학교 컴퓨터학부 부교수. 관심분야는 e-Science, 그리드시스템, PSE, 공개소프트웨어

허 대 영

2004년 국민대학교 컴퓨터학부(학사). 2005년 국민대학교 전산과학(석사). 2006년~ 국민대학교 전산과학 박사과정. 관심분야는 그리드 시스템, 시스템 아키텍쳐, 디자인 패턴