동적 부하 분산 시스템에서 효율적인 작업 크기 계산을 통한 성능 개선

최 민[†] · 김 남 기[††]

요 약

클러스터와 같은 분산 시스템에서 초기 작업 배치 시, 할당할 프로세스의 자원 요구량을 정확히 예측하여 작업을 분배할 수 있다면 보다 나은 시스템 성능을 얻을 수 있게 된다. 이 때 임의의 작업을 적절한 호스트에 배치하기 위해서 자원 기반 초기 작업 배치 (resource-based initial job placement) 기법은 그 작업의 자원 사용량을 미리 예측할 필요가 있다. 하지만 잘못된 자원 예측은 동적 부하 분산 시스템의 성능을 크게 떨어뜨리는 원인이 된다. 따라서 본 논문에서는 잘못된 예측에 의한 문제를 해결하기 위해 새로운 부하 기준을 제안한다. 새로운 부하 기준을 사용한 자원 기반 초기 작업 배치 기법은 프로세스의 유형에 관한 사전 지식 없이도 동작하는 장점을 가진다. 실험을 통해 본 논문은 동적 부하 분산 시스템에서 제안하는 방식이 기존의 방식에 비해 향상된 성능을 가짐을 보인다.

키워드 : 부하 분산, 클러스터 시스템, 분산 시스템 , 작업 스케줄링

Performance Improvement using Effective Task Size Calculation in Dynamic Load Balancing Systems

Min Choi[†] · Namgi Kim[††]

ABSTRACT

In distributed systems like cluster systems, in order to get more performance improvement, the initial task placement system precisely estimates and correctly assigns the resource requirement by the process. The resource-based initial job placement scheme needs the prediction of resource usage of a task in order to fit it to the most suitable hosts. However, the wrong prediction of resource usage causes serious performance degradation in dynamic load balancing systems. Therefore, in this paper, to resolve the problem due to the wrong prediction, we propose a new load metric. By the new load metric, the resource-based initial job placement scheme can work without priori knowledge about the type of process. Simulation results show that the dynamic load balancing system using the proposed approach achieves shorter execution times than the conventional approaches.

Key Words : Load balancing, Cluster system, Distributed system, Task scheduling

1. 서 론

최근 네트워크와 마이크로프로세서 기술에 관한 연구가 활발해지면서 독자적인 워크스테이션들을 고속의 네트워크(high speed network)로 연결한 클러스터 시스템이 등장하였다. 클러스터 시스템은 자원공유(resource sharing)가 쉽고, 상용(commodity) 계산 노드, 네트워크 장치 등과 같은 시스템 구성 모듈(module)을 추가함으로써 손쉽게 성능 향상을 얻을 수 있는 장점이 있다. 그러나 클러스터 시스템에서는 일부 노드가 다른 노드들에 비해 많은 부하가 집중되는 경향이 있다 [1]. 따라서 부하가 적은(lightly loaded) 노드들의 컴퓨팅 자원을 활용하여 부하 불균형을 해소하는 부하 분산 시스템(load balancing system)의 개발이 전체 시스템 성능 향상을 위해서 반드시 필요하다.

초기 작업 배치 방식을 사용하는 부하 분산 시스템은 노드별로 작업을 균등하게 배분함으로써 시스템 전체의 자원 활용률을 높인다. 그러나 시스템 전체는 물론이고 각 노드별 자원 활용률도 향상시킨다면 추가적인 성능향상을 기대할 수 있다 [2]. 따라서, 초기 작업 배치 시 작업의 자원 요구사항을 고려하여 작업을 배치하면 시스템의 전체 성능을 크게 향상시킬 수 있다. 이를 위해, 과거에 실행된 적이 있는 작업의 자원 요구 형태를 예측하는 방법에 관한 다양한 연구가 수행되었다 [3-6]. 그러나 이러한 방법들은 어디까지나 과거 데이터를 바탕으로 미래의 자원 요구 형태를 예측

※ 이 논문은 경기대학교 경기도지역협력연구센터(GRRC) 사업으로부터 연구비를 지원받았음
† 준 회 원 : 한국과학기술원 전산학과 박사과정
†† 정 회 원 : 경기대학교 컴퓨터과학과 교수(교신저자)
논문접수 : 2007년 7월 10일, 심사완료 : 2007년 11월 2일

하는 것이므로 부정확 할 수 있다는 단점이 존재하며, 잘못된 예측을 사용하여 작업 특성을 분류하고 이를 배치하는 경우 작업의 실행시간 증가에 따른 심각한 성능저하가 발생한다.

따라서, 본 논문에서는 예측을 통한 방법을 사용하지 않으면서도 초기 작업 배치에 있어 성능저하를 방지할 수 있는 새로운 방법을 제안한다. 이를 위해 노드의 부하 정보와 자원 활용률에 관한 정보를 동시에 포함하는 새로운 부하 척도(load metric)을 제안하고, 이것을 활용한 동적 부하 분산 시스템을 평가한다.

2. 관련 연구 및 연구 동기

동적 부하 분산 시스템은 작업을 실행하기 전에 전체 시스템에서 작업의 요구사항을 가장 잘 만족시킬 수 있는 노드를 찾아 그 작업을 할당하는 초기 작업 배치 (initial job placement) 방식과 부하의 불균형이 발생했을 때 부하가 큰 노드에서 실행중인 작업을 부하가 작은 노드로 이동하는 프로세스 이동 (process migration) 방식으로 나뉜다. 프로세스 이동 방식은 실행중인 작업의 상태를 이동하는데 요구되는 비용(cost)이 크고, 구현 복잡도가 높다. 따라서, 부하 분산을 위한 일반적인 방법으로는 초기에 작업 배치를 사용하고, 부하 불균형이 발생하는 경우에 한해 부수적으로 프로세스 이동을 사용한다 [7]. 결과적으로 프로세스 이동 기법보다는 초기 작업 배치 방식의 성능이 전체 시스템 성능에 보다 큰 영향을 주게 된다.

초기 작업 배치 시 작업의 자원 요구 형태를 예측하기 위한 방법으로는 과거 그 작업을 실행해 본 결과를 바탕으로 미래 자원 요구 형태를 예측하는 방법 [3], 통계적 접근을 사용하는 방법 [4], 작업에 필요한 자원 요구 예측을 사용자가 직접 제공하는 방법 [5], 작업 실행 초반에 나타나는 자원 요구 형태를 이용한 예측 방법 [6] 등이 있다.

과거 자료 기반 예측 방법 [3]은 작업의 자원 요구 형태를 예측하는 가장 대표적인 방법으로 현재에도 널리 활용되고 있다. 이 방법은 예전에 그 프로세스를 실행할 때 얻은 자원 사용에 관한 자료를 바탕으로 그 프로세스를 다시 실행할 때에도 과거와 유사한 자원 요구 형태를 보일 것이라는 가정을 사용한다. 예를 들면, 과거에 그 프로세스가 CPU의 사용량이 많았다면 이번 실행에서도 그 프로세스는 CPU 자원을 주로 사용할 것이라고 예측하는 방식이다. 그러나, 이러한 과거 자료 기반 예측 방법은 같은 명령어(작업 이름)가 같은 옵션(인수)으로 실행되는 경우에 한해서만 과거에 기록된 것과 동일한 작업으로 인식하기 때문에 예측이 가능한 경우가 많지 않으며, 비록 예측이 되었다 할지라도 그 정확성을 확신할 수 없는 단점이 있다 [8].

통계적 접근 방법 [4]에서는 어떤 주어진 프로그램에 대하여 그 작업의 CPU, I/O, Memory 자원의 활용 정도를 파악하는데 있어 통계적인 패턴인식 (statistical pattern-recognition-based) 방법을 사용한다. 통계적 접근 방법은 예측을 시행하기까지 주어진 UNIX 시스템에서 프로그램이 실행되는 동안 자원 사용에 관한 상태를 기록하고 분석하는 오프라인 (off-line) 부분과 얻어진 자료를 바탕으로 실제로 예측을 수행하는 온라인 (on-line) 부분으로 구성된다. 통계적 접근 방법은 어떤 작업(task)의 자원(resource) 사용 빈도에 대해 통계적 평균값을 사용하므로 구현하기 쉽다는 장점이 있다. 그러나 예측의 정확도가 통계에 사용되는 데이터의 양에 비례하고 예외적인 입력이 들어올 경우 올바른 예측을 하지 못한다는 단점을 가지고 있다.

사용자 추측 방법 [5]은 사용자가 실행하고자 하는 작업을 시스템에 제출(submit)할 때 그 작업에 대한 사용자 추측 정보를 함께 제공하는 방법이다. 따라서 운영체제가 더 이상 작업의 자원 사용 행태에 대한 정보를 기록하거나 모니터링 할 필요가 없다는 장점이 있다. 하지만 이 방법은 사용자가 일일이 실행하려는 작업이 I/O bound 작업인지 CPU bound 작업인지, 아니면 Memory bound 작업인지에 대해서 판단해야 하기 때문에 사용자의 실행 오버헤드가 증가하고 자동화 시스템으로 만들기 어렵다는 단점이 있다 [9].

작업 실행 초반에 나타나는 자원 요구 형태를 이용한 예측 방법 [6]은 작업 실행 초반에 나타나는 작업의 자원 요구 형태가 해당 작업의 전체 실행 동안 유지될 것이라는 가정을 기반으로 한다. 이 방법은 일단 작업을 실행시키고 처음 1초 동안(within one second) 수집한 데이터를 바탕으로 해당 작업을 원격 실행(remote execution)하는 것이 적절한지 판단한다. 작업의 실행 초반에 수집되는 데이터를 바탕으로 CPU 활용률, I/O 활용률, 메모리 및 파일에의 접근 빈도 등을 계산하여 원격 실행이 적합하다고 결정된 경우 자원 요구 형태 추적을 위해 실행중인 작업을 종료시키고 원격지 노드에서 작업을 새로 시작한다. 그러나, 이러한 방법은 작업의 자원 요구 형태를 파악하기 위한 처음 1초 동안의 작업 실행시간을 낭비하게 되는 단점이 존재한다 [10][11].

작업의 자원 요구 형태에 대한 선행 지식(priori knowledge)를 얻는 가장 보편적인 방법은 과거 데이터에 의한 예측 방법이다. 그러나, 이러한 예측 방법은 앞서 언급한 바와 같이 예측의 부정확성, 잘못된 예측으로 인한 성능 저하, 각 작업의 자원 요구 형태 정보를 저장하기 위한 시간적 공간적 비용 증가, 예측을 위한 검색 비용 증가 등의 문제점을 안고 있다. 따라서 본 논문에서는 작업의 자원 요구 형태에 대한 선행 지식을 필요로 하지 않으면서도 초기 작업 배치에 있어 작업이 잘못 배치되는 상황(worst case job placement)를 방지하는 유효 작업수를 이용한 동적 부하 분산 시스템을 제안한다.

3. 유효 작업수를 이용한 동적 부하 분산 시스템

3.1 유효 작업수 개념

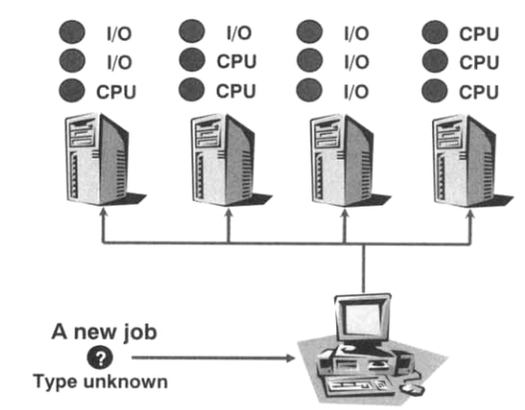
유효 작업수(number of effective tasks)란 시스템 성능에

실제로 영향을 미치는 프로세스의 수를 의미한다. 즉, 시스템 성능이 자원 활용률과 밀접한 관계가 있으므로 이를 고려한 부하 측정의 척도라 할 수 있다. 그림 1은 두 개의 작업(CPU bound, I/O bound)에 대해서 단독으로 실행하는 경우와 한 노드에서 동시에 두 개 작업을 실행하는 경우에 대해서 유효 작업수와 작업수(number of tasks)의 값을 나타낸 것이다. A와 B의 경우에 비해 D의 경우가 다른 점이라면 단지 시스템 자원을 최대한 활용하고 있다는 것과 약간의 실행시간 증가가 발생하였다는 것뿐이다. 따라서, 이러한 상황에서 시스템 부하를 2로 나타내는 작업수 부하 척도는 시스템 상태를 잘 반영하지 못한다고 할 수 있다. 반면, 시스템 부하를 1.08로 나타내는 유효 작업수는 "한 개 보다 조금 많은 작업이 실행 중이다" 는 것과 같으며, 시스템 자원 활용률을 고려하여 시스템 부하 수준을 잘 반영하고 있다고 볼 수 있다.

초기 작업 배치(initial job placement)을 사용한 부하 분산 시스템에 균등화된 자원 할당(balanced resource allocation)방식을 적용하기 위해서는 새로 배치할 작업의 자원 요구 형태를 미리 알 수 있어야 한다. (그림 2)는 클러스터 시스템의 각 노드에서 3개씩의 작업이 실행 중일 때, 새로운 작업을 배치하는 상황을 도시한 것이다. 그림 2에서 작업의 자원 요구 형태가 어떠한가에 따라서 그 작업을 실행하게 될 가장 최적의 노드가 결정된다. 그러나, 부하 척도로써 유효 작업수를 사용하는 경우 작업의 자원 요구 형태에 대한 선행 지식이 없더라도 그 작업이 잘못 배치되는 상황(worst case job placement)를 방지할 수 있다.

		Eff. # of tasks	# of tasks
A	I/O 94.77s CPU	1	1
B	I/O CPU 109.13s	1	1
C	I/O 218.24s CPU 218.34s	2	2
D	I/O 98.05s CPU 113.17s	1.08	2

(그림 1) 유효 작업수와 작업수의 예



(그림 2) 클러스터 시스템에서 새로운 작업 배치 구조 예제

(그림 3)은 (그림 2)와 같은 구조에서 새로운 작업이 배치되는 노드에 따라 발생할 수 있는 모든 경우의 수를 나열한 것이다. A는 한 노드에 동일한 자원을 요구하는 프로세스들이 모두 몰려있는 경우로 작업의 배치가 잘못된 경우(worst case job placement)이다. B는 각 노드에 각기 다른 자원을 요구하는 프로세스들이 균일하게 배치된 경우로 가장 이상적인 작업 배치(best case job placement)에 해당한다. C의 경우는 기존의 과거 자료 기반 예측 방법과 부하 척도로 작업수를 사용하여 새 작업을 배치할 때 발생 가능한 모든 경우를 나타낸다. 앞서 언급한 바와 같이 과거 자료 기반 예측 방법은 부정확성으로 인해 조합 가능한 모든 경우의 수가 발생할 수 있다. 마지막으로, D의 경우는 유효 작업수를 부하 척도로 사용할 때 발생 가능한 경우의 수이다. 자원 요구 형태에 대한 선행 지식과 관계 없이, 항상 유효 작업수가 작은 쪽으로 작업을 배치하기만 하면 작업이 가장 잘못 배치되는 상황을 피할 수 있음을 보여 준다. 그리고 이와 동시에 가장 이상적인 배치도 항상 발생할 수 있음을 보여준다. 유효 작업수를 사용하는 때에도 K의 경우처럼 최악의 배치는 아니지만 어느 정도 자원 사용에 불균형이 생기는 경우도 발생할 수 있다. 하지만, 이 경우에도 자원 요구 형태에 대한 선행 지식이 전혀 없는 상태에서 최악의 작업 배치를 피할 수 있다는 장점이 여전히 존재한다.

new job → already assigned jobs in a node	Case ID A	B	C	D	
CPU → CPU CPU CPU	√		√		
CPU → CPU CPU I/O			√	√	← K
CPU → CPU I/O I/O		√	√	√	
CPU → I/O I/O I/O			√		
I/O → I/O I/O I/O	√		√		
I/O → I/O I/O CPU			√	√	← K
I/O → I/O CPU CPU		√	√	√	
I/O → CPU CPU CPU			√		

A Worst case job placement
B Best case job placement
C Selection when the # of tasks & history used
D Selection when the eff. # of tasks used

(그림 3) 클러스터 시스템 예제에서 새로운 작업을 배치할 때 나타날 수 있는 경우의 수

3.2 유효 작업수 계산

유효 작업수는 식 (1)인 CPU bound 작업이 사용하는 평균 CPU 사용량에 대한 I/O bound 작업이 사용하는 평균 CPU 사용량의 비율에 기초한다. 즉, CPU bound 작업을 1로 보았을 때 I/O bound 작업이 사용하는 CPU 시간의 양이 차지하는 비율을 0부터 1사이의 값으로 표현한다.

$$\text{IO to CPU ratio} = \frac{\text{avg time consumed by IO jobs}}{\text{avg time consumed by CPU jobs}} \quad (1)$$

(그림 4)는 I/O to CPU 비율 값을 이용하여 유효 작업수를 계산하는 방법을 나타낸다. 그림에서 n(I/O)는 I/O bound 작업의 개수이고 n(CPU)는 CPU bound 작업의 개수이다. I/O to CPU ratio가 0.5보다 큰 경우는 I/O bound 작업이 사용하는 평균 CPU 시간이 CPU bound 작업이 사용하는 평균 CPU 시간의 반 이상에 해당한다. 이러한 경우는 굳이 별도의 계산을 통해 유효 작업수를 구하더라도 이를 부하 분산 시스템에 부하 척도로서 적용하였을 때 큰 효과를 볼 수 없기 때문에 유효 작업수를 기존 작업수와 동일하게 계산한다. 반면, I/O to CPU ratio가 0.5보다 작은 경우는 I/O bound 작업이 사용하는 평균 CPU 시간이 CPU bound 작업의 평균 CPU 시간의 반 이하에 해당한다. 이러한 경우는 CPU bound 작업 혹은 I/O bound 작업 중 개수가 많은 쪽에 해당하는 작업들이 시스템 성능에 더 큰 영향을 미친다. 다만, CPU bound 작업의 수가 I/O bound 작업의 수보다 많을 경우는 CPU bound 작업의 수에 I/O to CPU ratio를 더해준다. 이것은 CPU bound 작업을 1로 보았을 때 I/O bound 작업이 사용하는 CPU 시간을 부하 정도에 추가해 주기 위함이다. 반대로, I/O bound 작업의 수가 CPU bound 작업의 수보다 많을 경우는 I/O to CPU ratio를 더해주지 않는다. CPU bound 작업의 경우 I/O를 거의 사용하지 않기 때문에 I/O bound 작업을 1로 보는 경우 CPU 작업이 사용한 I/O 시간은 0에 가깝기 때문이다.

(그림 5)는 유효 작업수를 계산하는 세부 알고리즘이다. 이 알고리즘에서 I/O bound 작업과 CPU bound 작업의 구분은 한 epoch에서 소모하는 시간 할당량과 CPU factor를 이용한다. CPU factor란 사용자 CPU 시간이 전체 CPU 시간에서 차지하는 비율을 말한다. 일반적으로 CPU bound 작업은 한 epoch에서 자신에게 할당된 시간 할당량을 모두 소모하는 반면, I/O bound 작업은 그렇지 못한 특성을 갖는다. 또, CPU bound 작업의 경우 전체 시간에서 사용자 CPU 시간이 차지하는 비율이 크며 I/O bound 작업은 시스템 CPU 시간의 비율이 크다. 단, CPU factor의 경우는 한 노드에서 여러 개의 작업이 실행되는 경우 한 작업의 시스템 CPU 시간의 계산이 다른 작업의 영향을 받는 경우가 생긴다. 따라서, 작업의 성격을 분류하는데 있어 CPU factor와 소모한 시간 할당량의 비율을 함께 고려하도록 한다. 소모한 시간 할당량의 비율은 새로운 epoch이 시작될 때 해당 작업이 할당 받은 시간 할당량 전체에 대해서 그 epoch이 끝날 때까지 소모한 시간 할당량이 얼마만큼을 차지하는가를 의미한다. 예를 들어, 8만큼의 시간 할당량을 받은 작업이 해당 epoch이 끝날 때 할당 받은 시간 할당량 8을 모두 소모하였다면 이 작업에 대한 소모한 시간 할당량의 비율은 100이다. 반면, 8을 할당 받아 4만큼 소모한 경우는 이 작업의 소모한 시간 할당량의 비율은 50이다. 그리고 CPU factor는 작업이 소모한 사용자 CPU 시간이 전체 CPU 시간에서 어느 정도를 차지하는가를 의미한다. 이렇게 얻어진 두 값을 곱하여 임계치(threshold value) 1500보다 큰 경우 CPU bound 작업으로 간주하고 그렇지 않은 경우 I/O bound 작업으로 간주한다. 이 때, 임계치인 1500이라는 값은 여러 가지 CPU bound 작업과 I/O bound 작업을 실행해 본 뒤에 실험적으로 얻은 값이다.

```
if ( I/O to CPU ratio < 0.5 ) {
    if ( n(I/O) < n(CPU) )
        effective # of tasks
            = MAX( n(CPU), n(I/O) )
                + I/O to CPU ratio;
  else
      effective # of tasks
          = MAX(n(CPU), n(I/O));
}
else {
    effective # of tasks = # of tasks;
}
```

(그림 4) 유효 작업수 계산식

```
read_lock(&tasklist_lock)

for_each_task(p) {
  if ((p->state == TASK_RUNNING ||
    p->state == TASK_UNINTERRUPTIBLE))) {

    nr += 1;  // 작업수 계산
    CPU factor 계산;

    if (I/O bound 작업이면) {
      # of I/O tasks++;
      total CPU quantum used by I/O tasks
        += p->quantum_used;
    } else if (CPU bound 작업이면) {
      # of CPU tasks++;
      total CPU quantum used by CPU tasks
        += p->quantum_used;
    }
  }
}

// I/O bound 작업 및 CPU bound 작업이 최소한
// 하나씩은 있어야 유효작업수를 계산함.
// 예를들어, CPU bound 작업만 3개 있으면
// 기존의 작업수를 그냥 반환함
if (# of I/O tasks != 0 && # of CPU tasks != 0) {
  I/O to CPU ratio 계산;

  If (I/O to CPU ratio < 0.5) {
  If (# of I/O tasks > # of CPU tasks) {
      nr = # of I/O bound tasks;
  } else if (# of I/O bound tasks <
      # of CPU bound tasks) {
      nr = # of CPU bound tasks
      + I/O to CPU ratio;
  }
  }
// I/O to CPU ratio가 0.5보다 클땐
// nr(작업수)값 자동반환
}

read_unlock(&tasklist_lock);
return nr;
```

(그림 5) 유효 작업수 계산 알고리즘

4. 성능 평가

4.1 구현 환경

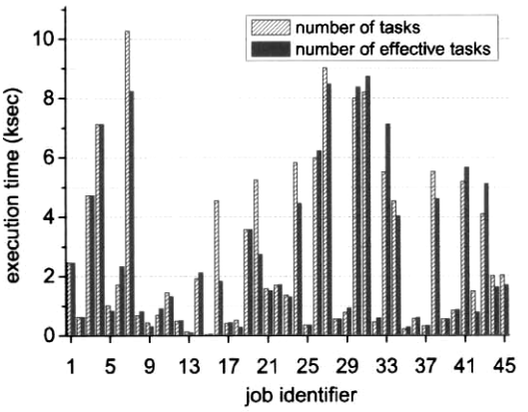
성능 평가를 위한 유효 작업수를 이용한 동적 부하 분산 시스템은 1대의 전역 작업 스케줄러(global job scheduler)와 3대의 연산 노드(computation node)로 구성된다. 유효 작업수의 구현은 리눅스 커널 2.4.2에 수정을 가하는 방식으로 구현되었으며 마이그레이션 데몬은 리눅스 사용자 수준(user-level)에서 개발되었다. 정보 매니저는 MFC 라이브러리를 활용하여 Visual C++로 개발된 윈도우 2000용 애플리케이션이다. 시스템 환경은 <표 1>과 같다.

성능 측정에 사용된 trace는 Cornell Theory Center(CTC)로부터 얻어진 데이터로 1996년 7월부터 1997년 5월까지 512노드 IBM SP2 머신의 작업 실행 정보를 기록한 로그에 해당한다. trace 데이터가 포함하고 있는 작업부하의 특성(workload characteristic)을 파악하기 위하여 실행시간(execution time), 도착 시간(arrival hour), 도착 시간 간격(inter-arrival time)을 측정하여 보았다. 그 결과 실험에 사용한 trace는 [12][13]에서 제시된 일반적인 유닉스 프로세스들이 갖는 특성과 크게 다르지 않았다. 따라서 위의 trace 데이터를 통한 자료 분석 결과는 일반적인 유닉스 프로세스들에 대해서도 일치하는 보편성을 가진다고 말할 수 있다.

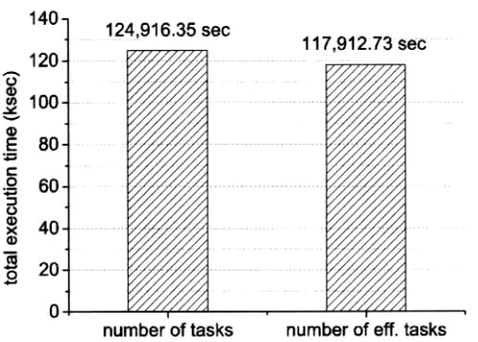
(그림 6)은 앞의 구현 환경에서 45개 작업에 대한 실행시간을 측정한 것이다. 작업의 자원 요구 형태를 예측하는 방법에서 사용하는 작업수 부하 척도와 본 논문에서 제안하는 유효 작업수 부하 척도를 사용한 경우의 실행시간을 비교하고 있다. 각각의 작업은 작업 식별자로 1부터 45까지의 값으로 표시하였다. 처음 4개 정도의 작업들에 대해서는 두 방식에서 성능변화가 전혀 없는 것을 알 수 있다. 이것은 개념이 서로 다른 자원을 요구하는 작업의 수가 최소한 두 개 이상 되어야 실질적인 유효 작업수를 통한 계산을 수행하기 때문이다. 그러나 7번, 16번, 20번 작업에서는 유효 작업수를 사용할 때 실행시간이 큰 감소하였다. 이는 예측을 통한 방법에서 잘못된 예측을 수행한 반면 유효 작업수를 사용한 방법에서는 그 작업이 실행되기에 가장 적절한 노드에 작업을 배치했기 때문이다. 반면, 31번, 33번, 41번 작업의 경우는 유효 작업수를 사용할 때 오히려 약간의 성능 저하가 발생했다. 이는 예측을 통한 방법이 이 경우에 한하여 최적화된 작업 배치를 수행한 반면 유효 작업수를 사용한

<표 1> 시스템 환경

전역 작업 스케줄러	CPU	Intel P4 1.8GHz
	MEM	256MB
	OS	Windows 2000 Professional
연산 노드	CPU	Intel P3 700MHz
	MEM	256MB
	OS	RedHat Linux 7.2, kernel 2.4.2



(그림 6) 작업 별 실행시간



(그림 7) 시스템 별 총 실행시간

방법에서는 최적화된 작업배치가 아니라 차선의 작업배치가 선택되었기 때문이다. 하지만 이 때에도 유효 작업수를 사용하는 방법은 최악의 작업배치를 방지하여 실행시간이 일정량 이상 늘어 나지 않는다.

(그림 7)은 각 작업들의 총 실행 시간의 합을 보여 주고 있다. 그림에서 알 수 있듯이, 유효 작업수를 사용한 동적 부하 분산 시스템은 예측 기반 시스템에 비해 약 6000초 가량의 전체 실행 시간(total execution time)단축을 가져왔으며 전체적으로 약 5.9%의 성능향상을 나타내었다.

5. 결론 및 향후 연구 과제

클러스터 시스템은 사용자에게 고성능과 확장성을 제공한다. 그러나, 일부 노드가 다른 노드들에 비해 많은 부하가 집중되는 부하 불균형 문제의 발생으로 인하여 이러한 장점을 저해하고 전체 시스템의 성능 저하로 연결된다. 이러한 문제를 해결하기 위해서 부하 분산 시스템(load balancing system)이 제안되었고 많은 연구가 진행되어 왔다. 그러나, 부하 분산 시스템에서 작업의 자원 요구 형태를 예측하여 초기 작업 배치를 하는 방식을 사용할 때, 예측이 잘못된 경우 작업의 실행시간 증가를 비롯한 성능저하 문제가 발생한다.

따라서 본 논문에서는 예측을 통한 방법을 사용하지 않으

면서도 초기 작업 배치에 있어 성능저하를 방지할 수 있도록 하는 유효작업수의 개념을 제안하였다. 유효 작업수는 실제로 시스템 성능에 영향을 미치는 작업의 수를 의미하며, 시스템 부하 정보와 자원 활용률에 대한 정보를 동시에 포함하는 개념이다. 유효 작업수를 사용한 동적 부하 분산 시스템을 설계 및 구현하고 그 성능을 측정해 본 결과 약 5.9%에 해당하는 성능향상을 가져왔다.

향후 연구과제로는 유효 작업수를 이용한 부하 분산 시스템 연구에서 노드 수를 증가시켜 알고리즘의 확장성 및 효율성을 검증해 보는 과정이 필요하다. 또한, 부하 분산 시스템에서 사용하는 위치 정책에 있어 자원 활용률을 고려하여 부하가 큰 노드들 사이의 작업 교환만으로 일정 수준의 부하 감소를 가져올 수 있도록 하는 방법을 연구할 계획이다.

참 고 문 헌

[1] M. Suzuki, et al., "A Task Migration Scheme for High Performance Real-Time Cluster System," Int. Conf. on Comp. and Their App., pp. 228 - 231, 2003.

[2] Y. Zhu, H. Jiang, X. Qin, and D. Feng, "Improving the Performance of I/O-Intensive Applications on Clusters of workstations," Cluster Computing, Vol. 9, No. 3, pp. 297-311, 2006.

[3] S. Krishnaswamy, S. W. Loke, and A. Zaslavsky, "Estimating Computation Times of Data- Intensive Applications," IEEE Distributed Systems Online, Vol. 5, No. 4, Apr. 2004.

[4] P. Kruger and R. Chawla, "The Stealth Distributed Scheduler," in Proc. of ICDCS, Jun. 1991.

[5] V. Subramani, et al., "Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests," IEEE HPDC, Jul. 2002.

[6] M. Schaar, K. Eye, L. Delcambre, and L. N. Bhuyan, "Load Balancing with Network Cooperation," In Proc. of ICDCS, Jun. 1991.

[7] K. P. Bubendorfer, "Resource Based Policies for Load Distribution," Master's thesis, Victoria Univ., 1996.

[8] K. Kurowsk, et al., "Multi-Criteria Grid Resource Management Using Performance Prediction Techniques," CoreGRID Integration Workshop, Nov. 2005.

[9] S. Elnikety, S. Dropsho, W. Zwaenepoel, "Tashkent+: Memory-Aware Load Balancing and Update Filtering in Replicated Databases," ACM SIGOPS Operating Systems Review, Vol. 41(3), Jun. 2007.

[10] X. Ren and R. Eigenmann, "Empirical Studies on the Behavior of Resource Availability in Fine-Grained Cycle Sharing Systems," In Proc. of ICPP, pp. 3-11, 2006.

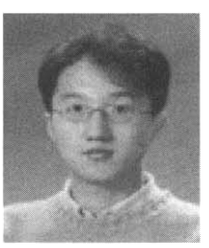
[11] M. H.-Balter, "Task Assignment with Unknown Duration," In Proceedings of the 20th International Conference on Distributed Computing Systems, April 2000.

[12] M. H.-Balter and A. B. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing," ACM Tran. on Computer Systems, Vol. 15, No. 3, pp. 253-285, Aug. 1997.

[13] W. E. Leland and T. J. Ott, "Load-balancing heuristics and process behavior," In Proc. of Performance and ACM Sigmetrics, Vol. 14, pp. 54-69, 1986.

최 민

e-mail : mchoi@camars.kaist.ac.kr
2001년 광운대 전자계산학과 학사
2003년 한국과학기술원 전산학과 석사
2003년~현재 한국과학기술원 전산학과 박사과정
관심분야 : 클러스터 시스템, 병렬 및 분산시스템, 컴퓨터구조



김 남 기

e-mail : ngkim@kyonggi.ac.kr
1997년 서강대 컴퓨터학과 학사
2000년 한국과학기술원 전산학과 석사
2005년 한국과학기술원 전산학과 박사
2005년~2007년 삼성전자 통신연구소 책임연구원
2007년~현재 경기대학교 컴퓨터과학과 교수
관심분야 : 통신 시스템, 네트워크