작업 이주시 보장/예약 기법을 이용한 프로세서 쓰레싱 빈도 감소

이 준 연† · 임 재 현††

요 약

동적 부하 분산 정책은 부하 분산을 결정하기 전에 각 노드는 현재 시스템 상태 정보를 수집한다. 이러한 방법에 기반한 부하 분산 정책은 예기치 못한 프로세서의 쓰레싱이 발생하게 된다. 본 논문에서는 부하 분산시의 프로세서 쓰레싱의 빈도를 감소시키기에 적합한 알고리즘을 제안하였다. 이 알고리즘은 다음의 세가지 요소로 구성되어 있다. 첫 번째는 송신 노드가 수신 노드에게 전송할 작업의 양을 결정하는 알고리즘이다. 두 번째는 송신 노드와 수신 노드 사이에서 전송할 작업의 크기에 관하여 상호 동의를 구하는 협상 프로토콜이다. 그리고 세 번째는 잠정적인 전송 대상을 선택하기 위한 대칭형 위치 정책의 설계이다. 본 논문에서 제안한 알고리즘의 성능 개선을 증명하기 위하여 Siman IV 시뮬레이션 도구를 이용한 모의실험을 통하여 제안된 알고리즘을 기존 대칭형 정책과 비교하여 평가하였다. 평균 응답 시간은 시스템이 저부하일 경우에는 기존 방법과 큰 차이를 보이지 않으나 평균 응답 시간이 급격히 증가하는 시스템 과부하일 경우에는 기존 연구에 비해 현저한 성능 증가를 보였다. CPU 오버헤드의 비율 또한 시스템의 과부하 상태에서는 본 논문에서 제안한 알고리즘이 기존의 알고리즘에 비해 더 낮다. 쓰레싱 발생 빈도는 본 논문에서 제안한 알고리즘이 수신 노드 쓰레싱 공통계수의 경우 시스템의 부하가 낮을 때 발생 빈도가 낮으며, 송신 노드 쓰레싱 공통계수는 시스템의 부하가 높을 때 더 유리하다. 따라서 시스템의 임계 큐의 상한값과 하한값을 적절하게 선택한다면 모든 시스템 상황에서 쓰레싱의 발생 빈도를 낮출 수 있다.

Reducing the frequency of processor thrashing using guarantee/reservation in process migration

Jun-Yeon Lee† · Jae-Hyun Lim††

ABSTRACT

In a dynamic load distribution policies, each node gathers the current system state information before making a decision on load balancing. Load balancing policies based on this strategy can suffer from processor thrashing. In this paper, we propose a new algorithm which attempts to decrease the frequency of the processor thrashing. the algorithm is based on the integration of three components. The first, the algorithm of which determine the size of jobs be transferred. The second, negotiation protocol which obtains a mutual agreement between a sender and a receiver on the transferring job size. And the third, a symmetrically-initiated location policy. The algorithm proposed in this paper used Siman IV as simulation tool to prove the improvement of performance. I analyzed the result of simulation, and compared with related works. The mean response time shows that there are no difference with existing policy, but appear a outstanding improvement in high load. The thrashing coefficient that shows the average response time, CPU overhead and the thrashing ratio at both the receiving and sending node has been used in the analysis. A significant improvement in the average response time and the CPU overhead ratio was detected using our algorithm when an overhead occurred in the system over other algorithm. The thrashing coefficient differed in the sending node and the receiving node of the system. Using our algorithm, the thrashing coefficient at the sending node showed more improvement when there was an overhead in the system, proving to be more useful. Therefore, it can be concluded that the thrashing ratio can be reduced by properly setting the maximum and minimum value of the system's threshold queue.

키워드 :

1. 서 론

일련의 프로세싱 노드들이 네트워크로 연결된 분산 컴퓨팅 시스템에서 일부 노드들은 다른 노드들에 비해 작업 도착율이 높은 경우가 자주 발생한다[1, 2]. 이러한 경우 부하의 불균형이 발생하며, 이 때 유휴 상태이거나 혹은 저부하 상태의 각 프로세서들의 사용율을 높이기 위하여 작업 부하를 분산시킴으로써 작업의 평균 응답 시간을 최소화하고,

† 정 회 원 : 동명정보대학교 멀티미디어공학과 교수
†† 정 회 원 : 공주대학교 영상정보공학부 교수
논문접수 : 2000년 7월 25일, 심사완료 : 2001년 1월 30일

CPU 사용률과 전체 시스템의 성능이 최대화될 수 있다. 부하 분산 정책은 정적과 동적으로 나뉘어지는데, 동적 부하 분산 정책은 시스템 상태의 변화를 동적으로 반영함으로써 정적 정책이나 혹은 부하 분산을 하지 않을 때에 비하여 현저한 성능의 개선을 제공한다. [3-7]에서 많은 동적 부하 분산 알고리즘이 제안되었으며, 또한 동적 부하 분산 알고리즘의 평가를 위한 접근법이 [8,9]에서 논의되었다. 동적 부하 분산 알고리즘에 적용되는 일반적인 메커니즘은 작업의 할당과 이주 정책이다. 작업 할당은 작업이 실행될 초기 노드의 위치를 의미한다. 작업 이주는 한 노드에서 다른 노드로 실행중인 작업의 동적 재배치를 의미한다.

동적 부하 분산 정책은 실행 위치에 따라 중앙 집중형과 분산 정책으로 구분할 수 있다. 중앙 집중형 정책에서는 부하 분산 결정을 하기 위한 전체 시스템 정보를 조정자가 소유/유지하게 되며, 거의 완벽한 부하 분산을 수행하게 된다. 단점은 조정자에 결함 발생시 성능 감소를 겪게 되며, 성능에 있어서 잠재적으로 병목 현상의 발생이 우려된다. 이러한 점에 관련하여 적절한 크기의 분산 시스템에서는 성능 병목 현상이 발생하지 않는다는 연구가 있었다[1]. 그러나 시스템들이 지역적으로 분산되어 있다면 조정자에게 문의하는 것은 매우 많은 비용이 소요되며, 이로 인하여 성능 문제를 발생시킬 것이다. 따라서 중앙 집중형 정책의 사용은 대규모의 분산 시스템에서는 부적절하다[13].

분산형 정책에서는 시스템 상태가 모든 노드로 분산되므로 중앙 집중형에서 발생하는 단점을 줄일 수 있다. 그러나 분산 정책은 각 노드의 상태 정보가 시스템의 모든 노드로 전송되어야 하므로 성능 문제를 일으킬 수 있다. 이에 대한 해결책으로는 무작위로 선택된 몇 개의 노드만을 샘플링함으로써 이러한 오버헤드를 줄일 수 있다[4, 7]. 분산형 정책에서 이보다 더 심각한 문제는 도착 시간 간격만큼이나 서비스 시간의 변화에 민감하다는 것이다[12]. 이러한 문제를 해결하기 위한 분산 정책은 부하 분산의 시작 위치에 따라 송신 노드 시작 정책과 수신 노드 시작 정책, 그리고 대칭형 정책의 세 가지가 있다.

송신 노드 시작 정책에서는 과부하 노드들이 저부하 노드들로 작업을 전송시키며, 수신 노드 시작 정책에서는 저부하 노드들이 전송될 수 있는 작업을 가진 과부하 노드를 찾는다. 일반적으로 시스템 부하가 상당히 낮을 경우에는 송신 노드 시작 정책이 수신 노드 시작 정책에 비해 더 좋은 성능을 나타내는 것처럼 보인다[4]. 그 이유는 시스템의 부하가 상당히 낮은 경우에는 저부하 노드를 찾는 확률이 과부하 노드를 찾는 확률보다 더 높기 때문이다. 이와는 반대로 시스템 부하가 상당히 높은 경우에는 과부하 노드를 찾기가 훨씬 더 쉽기 때문에 수신 노드 시작 정책이 더 유리하다[10, 14].

마지막으로 대칭형 정책은 송신 노드나 수신 노드가 모두 작업 전송의 대상을 찾기 위하여 알고리즘을 시작할 수 있는 방법이다[15]. 대칭형 알고리즘은 현재 시스템의 부하 상태에 따라 위치 정책에 가장 적합한 것이 될 것이다. 대칭형 알고리즘은 송신자 시작 정책과 수신자 시작 정책을 동적으로 변경할 수 있어야 한다. 다른 형태의 위치 정책에서 대칭형 알고리즘은 시스템의 성능과, 이러한 성능을 얻기 위하여 지불하여야 할 오버헤드를 고려할 때 가장 유리한 정책이다. 그러나 이 정책은 프로세서 쓰레싱을 감수하여야 한다[16].

프로세서 쓰레싱이란 여러 개의 노드가 하나의 대상 노드에게 동시에 작업을 전송하려 하거나, 혹은 부하 분산 협상의 대상이 되는 상황을 의미한다. 프로세서 쓰레싱이 수신 노드에게 과다한 작업이 전달되고, 따라서 수신 노드가 과부하 상태가 되며, 부하 분산의 감소로 인하여 동적 부하 분산 알고리즘의 성능이 저하될 수 있다.

따라서 본 논문에서는 대칭형 알고리즘에서 프로세서 쓰레싱을 교정할 수 있는 새로운 알고리즘을 제안하였다. 이 알고리즘은 작업을 전송할 노드가 수신 노드에게 미리 전송량을 보장받고, 수신 노드는 자신에게 전송될 양을 미리 예약하는 과정을 거치므로 이를 보장/예약 알고리즘이라 한다.

보장/예약 알고리즘은 다음의 세 가지 요소로 구성되어 있다. 첫 번째는 송신 노드가 수신 노드에게 전송할 작업의 양을 결정하는 알고리즘이다. 두 번째는 협상 프로토콜로, 이것은 송수신 노드 사이에서 전송할 작업의 크기를 전송하는데 상호 동의를 구하는 기능을 한다. 세 번째는 이러한 기능을 위하여 수정된 대칭형 정책이다.

본 논문은 다음과 같이 구성되어 있다. 제2장에서는 관련 연구를 소개한 후, 제3장에서 시스템의 모델과, 본 논문에서 제안한 보장/예약 알고리즘을 기술한다. 제4장에서는 시뮬레이션 모델과 비용 모델을 기술하고, 제5장에서 시뮬레이션의 결과를 분석하고, 마지막으로 제6장에서 결론을 맺는다.

2. 관련 연구

이 장에서는 본 논문과 관련된 연구를 요약하고자 한다.

2.1 Shivaratri와 Krueger의 부하 분산 정책

대칭형 위치 정책의 목적은 과부하 노드가 작업을 전송하려 할 경우에는 저부하 노드를 찾고, 저부하 노드가 작업을 수신하려 할 경우에는 송신 노드를 찾도록 하는 것이다[15]. 이 정책은 Shivaratri와 Krueger가 제안한 알고리즘이므로 이를 SK 알고리즘이라 한다.

2.1.1 자료 구조

임의의 노드 i에서 유지되는 정보는 세가지 리스트 구조로 구성되어 있다. 첫 번째는 송신 노드의 리스트인 $Slist_i$

로 이것은 작업을 전송할 잠정적인 송신 노드일 경우 그 노드의 *ID*를 포함한다. 두 번째는 수신 노드 $Rlist_i$로 이것은 잠정적인 수신 노드의 *ID*를 가지고 있다. 마지막으로 $OKlist_i$로 이것은 송신 노드도 아니며 수신 노드도 아닌 즉, 적정 부하를 가진 노드의 *ID*를 가지고 있다.

초기에 모든 노드는 유휴 상태로 간주하며 따라서 노드 *i*의 각 리스트의 초기값은 다음과 같다.

$$Slist_i = i+1, i+2, \cdots, n, 1, \cdots, i-1$$
$$Rlist_i = null$$
$$OKlist_i = null$$

2.1.2 송신 노드의 협상 시작

송신 노드가 협상을 시작한 경우 송신 노드가 수신 노드를 찾기 위하여 수신 노드가 기록된 $Rlist_i$의 헤더를 검사하게 된다. 만약 검사된 노드가 현재 수신 노드 상태를 유지한다면, 이 노드로 작업 부하를 전송한다. 그러나 이 노드의 상태가 변경되었다면, $Rlist_i$에서 노드의 *ID*를 제거하고, 해당 노드의 상태에 따라 적절한 리스트에 추가된다. 이를 알고리즘으로 표현하면 (알고리즘 1)과 같다.

```
1. probe the head of Rlist_i, say j, to determine whether j is
   receiver
2. If j = receiver
      then { return j to the transfer policy
             STOP. }
      else { remove j from Rlist_i
             if j = sender then add j to the head of Slist_i
                           else add j to the head of OKlist_i
             }
3. If Rlist_i is empty
     then  {          negotiation STOP
                      return FAILURE to transfer policy
           }
           else go to step 1.
```

(알고리즘 1) SK 송신노드 시작 알고리즘 - 송신노드측

이 때 송신 노드 *i*로부터 탐지 메시지를 받은 노드 *j*가 수행하는 과정은 (알고리즘 2)와 같다. 먼저 노드 *i*로부터 송신 협상 탐지 메시지를 받았다는 것은 이 노드의 상태가 송신 상태가 되었다는 것을 의미한다. 따라서 현재의 리스트에서 *i*를 제거하고, 노드 ID를 $Slist_i$의 헤더에 추가한다. 비록 노드 *i*가 현재 $Slist_i$에 위치하더라도 현재의 위치에서 삭제되는 이유는 리스트의 헤더에 가까울수록 더 최신의 정보를 포함하고 있기 때문에 현재의 리스트 위치에서 먼저 제거된 후, 송신 리스트 $Slist_i$의 헤더에 추가된다.

```
1. remove i from whatever list it is in.
2. add it to the head of Slist_i
3. Send reply message to i indicating whether j is a
   recerver, sender, or OK.
```

(알고리즘 2) SK 송신노드 시작 알고리즘 - 수신노드측

2.1.3 수신 노드 시작 협상

수신 노드가 협상을 시작한 경우, 수신 노드가 송신 노드를 찾게 된다. 위치 정책의 수행 과정은 다음과 같다.

먼저 송신 노드를 찾기 위하여 송신 노드 리스트 $Slist_i$의 헤더에 위치하는 노드 ID를 검사한다. 이 때 $Slist_i$의 요소가 없다면, $OKlist_i$의 마지막 노드부터 검사하게 된다. 그 이유는 노드의 *ID*가 리스트의 마지막에 가까울수록 기록된 상태 정보가 오래 되어 상태가 바뀔 확률이 높기 때문이다. $OKlist_i$에서 송신 노드를 찾지 못한다면, $Rlist_i$의 마지막 노드에서부터 송신 노드인가를 검사한다. 송신 노드를 찾았다면, 이 노드의 ID를 현재의 리스트에서 제거한 후 $Slist_i$의 헤더에 추가한 후, 노드 ID를 돌려준다. 그러나 찾아진 송신 노드가 작업을 이주한 후 OK 상태가 되었다면 이는 송신 노드 리스트에 추가하는 것이 아니라 $OKlist_i$에 추가하여야 한다.

탐지된 노드가 송신 노드가 아니라면 이 노드의 현재 상태를 파악하여 적절한 리스트에 추가되어야 한다. 이 과정을 다음 (알고리즘 3)과 (알고리즘 4)에서 기술하고 있다.

```
1. probe a selected node, say j, to determine whether j is a
   sender :
      choose the head of Slist_i
      if Slist_i is empty, then choose last node from OKlist_i
      if both Slist_i and OKlist_i are empty,
            choose the last node from Rlist_i
2. If j responds that it is a sender
            then {
                  add j to the head of Slist_i
                  return j to the transfer policy
                  STOP
                 }
   If j responds that it is a sender, but will no longer be
     a sender after migrating a task
            then  {
                  remove j from whatever list it is in,
                  add j to the head of OKlist_i ,
                  return j to the transfer policy
                  STOP
                  }
```

(알고리즘 3) SK 수신 노드 시작 알고리즘 - 수신노드측

```
If j responds that it is not a sender
        then     {
                 remove j from whatever list it is in
                 if j = receiver
                       then add j to the head Rlist_i
                       else add j to the head of OKlist_i
                 }
3. If all nodes have been proved
        then STOP negotiation and return FAILURE
        else go to 1.
```

(알고리즘 4) SK 수신 노드 시작 알고리즘 - 수신노드측(계속)

수신 노드로부터 탐지 메시지를 받은 노드 *j*는 다음 과정을 수행한다. 만약 자신의 상태가 송신 노드 상태가 아니라

면 이를 메시지를 전송한 노드에게 알리고, 메시지 전송 노드의 ID를 $Rlist_j$에 추가한다.

반면에 자신이 송신 노드 상태라면, 하나의 작업을 이주시킨 후 자신의 상태를 노드 i에게 알린다.

```
1. If j is not a sender
     then    {
             return informing message
             remove i from whatever list it is in
             add i to the head of Rlist_j
             }
2. If j is a sender then migrate a task to i
   Inform i of j's state after the migration
```

(알고리즘 5) SK 수신 노드 시작 알고리즘 - 송신노드측

본 논문에서 사용하고자 하는 위치 정책은 이 절에서 언급한 대칭형 정책이다. 위의 알고리즘에서 기술한 바와 같이 대칭형 정책은 자신이 과부하 노드가 되었을 경우 송신 노드로 인식되어 작업을 전송하기 위하여 송신 노드 협상을 수행하며, 저부하 노드가 되었을 경우 수신 노드가 되어 수신 노드 협상을 수행하게 된다. 만약 시스템의 각 노드의 작업이 편중되어 발생하는 경우 OK 상태를 제외한 노드의 개수를 n이라 할 때 위치 정책은 $n \cdot (n-1)$개의 탐지 메시지를 발생시키게 되며, 또한 이러한 메시지의 처리를 위하여 CPU의 연산 능력을 낭비하게 된다. 또한 각 노드가 모두 협상에 참여하게 되므로 예기치 못한 CPU의 쓰레싱 현상이 발생하게 된다.

그리고 작업의 과부하 혹은 저부하의 정도가 매우 심한 경우 작업을 이주한 후에도 계속하여 과부하 혹은 저부하의 상태를 유지하게 된다.

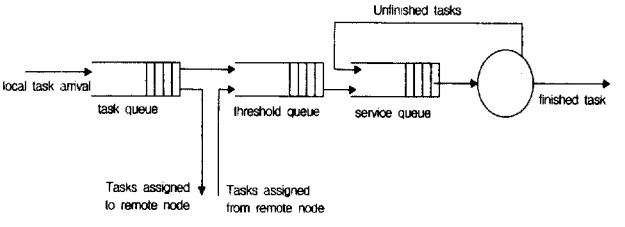
따라서 본 논문에서는 이러한 현상을 방지하기 위하여 작업의 이주시 OK 상태를 보장받을 수 있을 때까지 작업의 이주를 계속하며, 또한 작업의 송수신 협상에 참여한 노드는 더 이상의 메시지를 받지 않음으로써 불필요한 메시지 전송 및 처리를 방지하기 위한 보장/예약 알고리즘을 제안하고자 한다.

3. 시스템 모델 및 알고리즘

3.1 시스템 모델

본 논문에서 사용하는 시스템 모델은 각 노드는 다른 모든 노드로 도달할 수 있다. 본 논문에서는 작업이 도착하는 노드를 도착 노드라 하고, 작업이 실행되는 노드를 실행 노드라 한다. 일반적으로 작업 할당 알고리즘은 도착 노드에서 실행 노드로 작업을 할당하는 것을 의미한다. 실행 노드가 도착 노드와 동일한 경우를 로컬 할당이라 하고, 그렇지 않으면 원격할당이라 한다.

각 처리 노드는 작업 큐, 서비스 큐, 그리고 임계 큐 등 3개의 FIFO 큐로 구성되어 있다. 작업 큐와 임계 큐는 크기가 제한되어 있지 않다. 서비스 큐는 그 크기가 제한되어 있고, 최대 Q_0개의 작업만을 가질 수 있다. 로컬 작업이 도착하면, 이는 태스크 큐로 들어가서 작업 할당을 위한 후보가 된다. 작업은 자신의 임계 큐로 들어가는 로컬 할당이 되거나 혹은 다른 원격 노드의 임계 큐로 들어가는 원격 할당이 될 수 있다. 일단 작업이 도착하면 이는 도착한 노드에 로컬로 할당되는 다른 알고리즘[4, 11, 13]들과는 달리 본 논문에서는 작업을 로컬 할당할 것인지 원격 할당할 것인지를 결정할 때까지 작업 큐에 머무르게 한다. 이렇게 함으로써 각 노드를 제어할 때 훨씬 더 융통성을 발휘할 수 있다. 작업이 임계 큐에 들어오게 되면 다른 노드로 재할당될 수 없다. 즉 작업 재할당을 허용하지 않는다. 이것은 작업을 연속적인 이동을 막는 중요한 역할을 한다. 따라서 불필요한 CPU와 통신 오버헤드를 줄일 수 있다.



(그림 1) 프로세싱 노드 모델

임계 큐에 있는 작업은 디스패치될 때까지 실행되지 않는다. 임계 큐에 있는 작업은 서비스 큐의 내용이 일정 크기 이하가 될 때 디스패치된다. 서비스 큐의 작업은 라운드-로빈 방법으로 처리된다. 작업은 정해진 시간 T_{CPU}만큼 CPU를 점유할 수 있고, 주어진 시간 간격동안 처리가 끝나지 않으면 서비스 큐의 마지막으로 진입하여 다음 CPU 사용 시간을 기다리게 된다[17].

3.2 보장/예약 알고리즘

3.2.1 보장/예약 프로토콜

보장/예약 프로토콜은 프로세서 쓰래싱을 해결하기 위하여 고안된 것으로 기본적인 방법은 다음과 같다. 송신 노드와 수신 노드의 쌍이 결정되면 송신 노드는 폴링 메시지에서 수신 노드로의 전송을 보장할 수 있는 작업의 수(보장값 : Guarantee value)를 정의한다. 이 값을 근거로 수신 노드는 송신 노드로부터 받아들일 작업의 수(예약값 : Reservation value)를 결정한다. 예약값은 송신 노드로부터 약속된 양만큼 받기 위한 할당량으로서, 그리고 다른 송신 노드로부터의 작업 송신을 막기 위하여 사용된다[17].

바꾸어 말하면, 수신 노드는 예약값만큼 자신의 부하를 증가시킴으로써 다른 잠정적인 노드와의 협상 요청을 막고, 이미 작업의 전송이 시작된 것처럼 보장하기 위하여 예약값을 사용한다. 따라서 수신 노드는 다른 송신 노드가 작업

전송을 요청할 경우, 이미 송신 노드가 작업군을 전송중인 것처럼 동작한다. 작업군의 전송이 완료되면, 이에 상응하는 할당량을 증가시킨다. 물론 새로운 부하 상태를 측정할 때 수신 노드는 새로 도착한 원격 작업을 계산한다. 예약/보장 프로토콜은 각 노드에서 두가지 속성을 사용한다.

정의 1 : 노드 P_i의 총 예약값(RES_i)은 모든 송신 노드로부터 자신에게 예약된 작업의 수이다.

정의 2 : 노드 P_i의 총 보장값(GUR_i)은 모든 수신 노드가 전송을 보장한 작업의 수이다.

3.2.2 작업부하 상태 측정

보장값과 예약값, 그리고 수행되는 모든 작업의 수를 근거로 하여 노드P_i의 부하를 다음과 같이 정의할 수 있다.

정의 3 : 노드 P_i의 실제 부하(K_i)는 작업 큐, 서비스 큐, 그리고 임계 큐에 존재하는 작업의 총 수를 의미한다.

정의 4 : 노드 P_i의 가상 부하 $VW_i = K_i + RES_i - GUR_i$ 이다.

부하를 정확하게 측정하기 위하여 여러 가지 부하 요소를 고려하는 것은 간단한 부하 인덱스에 비해 현저한 성능 개선을 가져오지는 못한다[4, 5]. 따라서 본 논문에서는 각 노드의 처리 용량의 수준을 나타내기 위하여 3단계의 부하 측정 기법을 사용한다. 이 3단계는 H-load(과부하), N-load(적정 부하), L-load(저부하 상태)이다. 한 노드의 부하 상태는 가상 부하 VW_i에 의하여 결정된다. <표 1>은 부하 상태와 가상 부하 VW_i의 관계를 나타낸다.

〈표 1〉 VW_i를 기준으로 한 부하 상태

부하 상태	기 준
L-load	$VW_i \leq lower\_threshold$
N-load	$lower\_thershold < VW_i \leq upper\_threshold$
H-load	$VW_i > upper\_threshold$

여기서 *lower_threshold*와 *upper_threshold*는 시스템 설계시 정해지는 매개변수로서, 이 값들의 선택은 동적 부하 균등화 알고리즘의 성능에 지대한 영향을 미친다.

3.2.3 위치 정책

보장/예약 알고리즘은 [12]에서 제안된 적응성있는 대칭형 위치정책을 사용한다. 이 정책의 특징은 다른 노드의 최근 부하 상태를 탐지하기 위하여 폴링시에 수집된 부하 정보를 사용하는 것이다. 즉, 폴링 메시지의 종류에 따라 자신이 보유하고 있는 각 노드의 상태 정보를 수정함으로써 별도의 비용을 지불하지 않고 최신의 부하 정보를 유지할 수 있다.

노드 P_i의는 부하 정보를 위하여 3개의 리스트 구조를 유지한다. 첫 번째 $SList_i$는 잠정적으로 송신 노드로 간주된 모든 노드의 *id*를 포함한다. 두 번째, $RList_i$는 잠정적으로 수신 노드로 간주된 모든 노드의 *id*를 포함하며, 세 번째로 $NList_i$는 적정 부하를 가진 노드의 *id*를 포함한다.

각 리스트의 헤드는 가장 최근의 부하 정보의 노드 *id*를 가리킨다. 따라서 송신 노드는 $RList_i$의 헤드가 가리키는 노드를 작업 전송을 위한 잠정적인 수신 노드로 선택한다. 이와 마찬가지로, 수신 노드는 *SList*의 헤드가 가리키는 노드를 잠정적인 송신 노드로 선택한다.

정책이 적응성이 있다는 것은 전반적인 시스템의 부하가 높을 때는 수신 노드 시작 정책을 사용하고, 시스템의 부하가 낮을 경우에는 송신 노드 시작 정책을 사용하듯이, 시스템의 상태에 따라 적용되는 알고리즘이 동적으로 변할 수 있는 정책을 말한다. 이러한 정책은 Eager. et. al.의 연구 결과와 일치한다[13]. 한 노드의 부하 전송의 협상은 노드의 부하 상태가 변경되는 순간에 결정된다. 송신 노드의 협상은 지역 작업의 도착으로 인하여 그 노드의 상태가 H-load상태가 되는 순간에 시작된다. 마찬가지로 수신 노드의 협상은 한 작업의 완료로 노드의 상태가 L-load 상태가 되는 순간에 시작된다.

폴링 기반 부하 균등화 알고리즘에서 무제한의 폴링이 CPU 시간과 네트워크 대역폭을 낭비할 수 있다. 따라서 본 논문에서 제안한 알고리즘에서는 부하 균등화 과정에서 만들어질 수 있는 폴링의 횟수에 제한을 둔다. 이 제한을 *probe_limit*라 한다. 또한 *probe_limit*의 사용은 연속적인 폴링으로 인한 무기한 연기를 막을 수 있고, 따라서 네트워크에서의 불필요한 충돌도 방지할 수 있다.

3.2.4 전송량 결정

적절한 전송량의 크기는 알고리즘의 성능에 따라 결정되기 때문에, 송신 노드와 수신 노드 사이의 상호 동의를 구하기 위하여 세가지 규칙을 정하여 사용한다. 전송량의 협상은 보장/예약 프로토콜에 포함하여 실행할 수 있다. *max*를 수신 노드가 송신 노드로부터 받아들이고자 하는 최대 작업수라 하고, t를 송신 노드가 수신 노드로 전송하고자 하는 작업의 수라 한다.

가정 1 : 만약 수신 노드가 *max*개의 작업을 수신하였다면, H-load 상태가 되지 않아야 하고, 작업 전송의 협상과 작업 전송중에 새로운 작업의 도착이나 전송을 무시한다.

가정 2 : 만약 수신 노드에서 t개의 작업이 제거되었다면, 송신 노드는 L-load 상태로 되지 않아야 한다.

가정 3 : 만약 t개의 작업이 전송되었다면, 수신 노드에서 기대되는 효율은 송신 노드의 부하 효율보다 크지 않아야 하고, 협상중의 새로운 작업의 도착과 출발 및 작업 전송 지연을 무시한다.

여기서 *max*와 *t*의 값은 *MaxAssign()*과 *NumAssign()* 두가지 함수에 의하여 결정된다. *MaxAssign()* 함수는 수신 노드 P_r에서 자신이 전송받을 수 있는 최대 작업 부하 *max*를 구하기 위하여 사용되고, *NumAssign()* 함수는 송신 노드 P_s에서 전송할 수 있는 작업 부하 *t*를 구하기 위하여 사용된다.

이 두 함수는 다음과 같이 정의될 수 있다.

VW_r은 수신 노드 P_r의 현재 가상 부하라 하고,
VW'_r은 작업 전송후 예측되는 수신노드 P_r의 가상 부하라 한다.
P_r이 *max*개의 작업을 받았다면

$$VW'r \approx VWr + max$$

가정 1에 의하여

$$VW'r \leq upper\_threshold$$

따라서

$$max \leq upper\_threshold - VWr$$

*max*의 가능한 최대값은

$$max = upper\_threshold - VWr \quad (1)$$

VW_s를 송신 노드 P_s의 현재 가상 부하라 할 때
가정 2에 의하여

$$VW_s - t > lower\_threshold$$
$$t < VW_s - lower\_threshold \quad (2)$$

공식 (1)을 사용하여 송신 노드 P_s가 예측하는 P_r의 가상 부하VW_r"는 다음과 같다.

$$VWr'' \approx upper\_threshold - max \quad (3)$$

식 (3)은 가정 3에 의하여

$$VWr'' + t \leq VWs - t$$

와 같이 표현될 수 있다.
따라서

$$t \leq \frac{VW_s + max - upper\_threshold}{2} \quad (4)$$

여기서

$$t \leq max \quad (5)$$

그러므로 *t*의 값은 (2), (4), (5)를 만족하는 최대 정수값이다.

*MaxAssign()*은 수신 노드 P_r이 예약값 RES_r을 수정하기 전에 자신이 전송받을 수 있는 최대 부하량을 연산하기 위하여 호출되며, *NumAssign()*은 송신 노드 P_s에 의하여 보장값 GUR_s가 수정된 후에 호출된다.

전송할 작업의 크기를 결정하기 위하여 송신 노드 P_s는 폴링 메시지에서 보장값 *gur*을 선언한다. 수신 노드 P_r는 폴링 메시지를 받은 후 *MaxAssign()*함수를 호출함으로써 *max*값을 결정한다.

예약값 *res*는 *max*와 *gur*중 더 작은 값으로 결정된다. *ACK* 메시지로서 수신 노드로부터 송신 노드로 *max*값과 *res*값이 전송된다. 그 후 전송할 작업량의 크기인 *t*값을 결정하기 위하여 송신 노드는 *NumAssign()*함수를 호출한다.

작업군이 수신 노드로 전송된 후, 송신 노드는 *Receiver NewVW()* 함수를 호출하여 수신 노드의 새로운 가상 부하를 평가한다. 이러한 평가는 *max*의 값과 전송할 작업군의 크기 *b*의 값을 기초로 하여 결정된다. *b*의 최종값은 *t*와 *res*중 더 작은 값이 선택된다.

VW''_r을 P_r의 가상 부하라 하고,
VW'''_r을 *b*만큼의 작업을 받아들인 후의 P_r의 새로운 가상 부하라 할 때

$$VW'''_r = VW''_r + b$$

여기서 공식 (3)에 의하여

$$VW'''_r \approx upper\_threshold - max$$

따라서

$$VW'''_r = upper\_threshold - max + b$$

(알고리즘 6) ReceiverNewVW() 함수

수신 노드의 가상 부하의 평가는 송신 노드의 리스트 구조를 유지하는데 사용된다.

3.2.5 송신 노드 협상 시작 알고리즘

분산 시스템의 각 노드들의 평균 부하가 정해진 임계값의 하한선 *lower_threshold*보다 더 작은 경우에는 수신 노드가 송신 노드를 찾을 확률보다 송신 노드가 수신 노드를 찾을 확률이 더 높아지며, 또한 협상의 대상 노드를 찾는데 소요되는 메시지의 수도 현저하게 감소된다. 이 경우 대칭형 위치 정책은 송신 노드가 협상을 시작하여야 한다.

송신 노드를 P_s라 하고, 수신 노드를 P_r이라 할 때, 보장값 *gur*과 예약값 *res*를 고려한 송신 노드의 협상을 시작하는 알고리즘은 다음 (알고리즘 7)과 같다. 먼저 협상을 시작하는 송신 노드는 폴링 메시지를 전송할 수신 노드를 선택하기 위하여 자신의 수신 리스트 $RLists_s$의 첫 번째 노드를 선택한다. 이렇게 선택된 노드에게 폴링 메시지를 전송하기 위하여 자신의 보장값 *gur*을 계산한 후, 이를 폴링 메시지에 포함하여 대상 노드에게 전송한다.

폴링 메시지 : *POLLING(gur)*

Procedure *S1*
1. select target node :
   target node P_r = head of $RList_s$
2. determine guarantee value *gur* :
   gur = (number of tasks in task queue) - GUR_s
3. $GUR_s = GUR_s + gur$
4. $VW_s = K_s + RES_s - GUR_s$
5. send a polling message to P_r

(알고리즘 7) 송신 노드의 협상 시작 알고리즘

송신 노드 P_s로부터 폴링 메시지 *POLLING(gur)*를 받은 잠정적인 수신 노드 P_r은 먼저 P_s의 정보가 포함된 리스트를 수정한 후, 자신의 가상 부하 상태를 검사하게 된다. 자신이 수신 노드의 상태가 아니라면 *NACK* 메시지를 발생시키는 *R1'*((알고리즘 11))으로 분기하고, 수신 노드의 상태라면 자신이 전송 받을 수 있는 작업의 양을 알리는 예약값 *res*와 현재의 가상 부하를 계산한 후, 응답 메시지 *ACK(gur, max, res)*를 송신 노드에게 전송한다. 이 과정을 다음 (알고리즘 8)에서 기술한다.

Procedure *R1*
1. List update
   Remove P_s from whatever list it is in
   add it to the head of $SList_r$.
2. If P_r is NOT a receiver(i.e. $VW_r \neq$ L-load)
   goto procedure *R1'*
   endif
3. Determine reservation value *res* :
   max = *MaxAssign()*
   if *max* $\leq$ *gur* then *res* = *max*
   else *res* = *gur*
4. Accumulate reservation value *res* :
   $RES_r = RES_r + res$
5. $VW_r = VW_r + RES_r - GUR_r$
6. Send an *ACK* message to P_s

(알고리즘 8) 수신 노드의 예약값 계산 알고리즘

P_r로부터 *ACK(gur, max, res)*을 받은 P_s는 수신 노드로부터 받은 예약값 *gur*과 자신이 결정한 보장값 *res*를 비교하여 전송할 작업의 크기를 결정한다. 그리고, 작업을 전송한 후에 달라질 수신 노드의 부하 상태를 수정하고, 자신의 보장값 *GUR*과 예약값 *RES*을 수정한 후, 작업을 전송하고 수행을 종료한다. 이 과정이 다음 (알고리즘 9)에 나타나 있다.

Procedure S2
1. Release guarantee value *gur* :
   $GUR_s = GUR_s - gur$
2. Determine job size *b* :
   t = *NumAssign()*
   if $t \leq res$ then $b = t$
   else $b = res$
3. Select *b* tasks from task queue :
   If no task can be selected
   goto procedure *S2'*
   endif
4. Transfer *b* tasks to P_r :
   $K_s = K_s - b$
5. $VW_s = K_s + RES_s - GUR_s$
6. List update :
   Estimate P_r's new effective load state by calling *ReceiverNewVW()*.
   Move P_r to the head of appropriate list of P_s.
7. Reset *probe_limit* counter and stop

(알고리즘 9) 송신 노드의 작업 전송량 결정 알고리즘

송신 노드 P_s로부터 폴링 메시지 *TRANSFER(tasks, b, res, VW_s)*를 받은 P_r는 송신 노드의 가상 부하 상태 VW_s에 따라 리스트를 수정하고, 전송받은 작업을 임계 큐에 추가한 후, 예약값과 자신의 가상 부하를 재설정한다.

Procedure R2
1. Lists update :
   Move P_s to the head of the appropriate list according to VW_s
2. Append *b* tasks onto the threshold queue :
   $K_r = K_r + b$
3. Release reservation value *res* :
   $RES_r = RES_r - res$
4. $VW_r = K_r + RES_r - GUR_r$

(알고리즘 10) 수신 노드의 예약값과 가상부하 재설정

송신 노드 P_S로부터 협상을 시작하는 폴링 메시지 *POLLING(gur)*를 받았으나 자신의 상태가 더 이상 저부하 상태(수신 가능 상태)가 아닌 경우 프로시저 *R1*에서 프로시저 *R1'*으로 분기된다. 이 때 수행되는 과정이 다음 (알고리즘 11)이다.

Procedure *R1'*
1. Send a *NACK* message to P_s,
   maintain VW_r

(알고리즘 11) 수신 노드가 수신 상태가 아닌 경우의 알고리즘

프로시저 *R1'*으로부터 전송된 폴링 메시지 *NACK(gur, VW_r)*를 받은 송신 노드 P_s는 현재의 잠정적인 수신 노드가 수신 상태가 아님을 감지하고, 현재의 P_r 노드의 *id*를 $RList_s$로부터 제거한 후 VW_r 상태에 따라 적절한 리스트에 추가한 후, 잠정적인 수신 노드를 다시 선택하기 위하여 프로시저 *P1*을 다시 시작한다.

Procedure *S3*
1. Lists update :
   Move P_r to the head of the appropriate list according to VW_r
2. Release guarantee value *gur* :
   $GUR_s = GUR_s - gur$
3. $VW_s = K_s + RES_s - GUR_s$
4. Initiate another polling session by going to *S1* unless either :

a. *probe_limit* is exceeded or
b. $RList_s$ is empty or
c. P_s is no longer a send (i.e. $VW_s \neq$ H-load)

(알고리즘 12) 잠정적인 수신 노드의 재설정 알고리즘

한편 프로시저 *S2*에서 *NumAssign()* 함수의 실행 결과와 예약값의 크기를 기초로 하여 전송할 작업의 크기가 정해졌으나, 작업 큐에서 실제로 이 크기만큼의 작업을 선택할 수 없는 경우가 발생할 수 있다. 이 경우 실행되는 과정은 (알고리즘 13)와 같다.

Procedure *S2'*
1. $VW_s = VW_s + RES_s - GUR_s$
2. List update :
   Estimate P_r's new virtual load by calling *ReceiverNewVW()*.
   Move P_r to the head of the appropriate list of P_s accordingly.
3. Send a *NACK* message to P_r,
   maintain VW_s
4. Initiate another polling session by going to *S1* unless either
   a. *probe_limit* is exceeded or
   b. $RList_s$ is empty or
   c. P_s is no longer a sender (i.e. $VW_s \neq$ H-load)

(알고리즘 13) 송신 노드에서 전송할 작업의 선택이 불가능한 경우

프로시저 S2'에서 송신 노드 P_s가 자신이 보장한 값만큼의 작업을 전송할 수 없을 경우, 수신 노드 P_r는 폴링 메시지 *NACK(res, VWs)*를 받는다. 이 경우 P_r는 자신에게 예약되었던 크기의 작업 부하가 전송될 수 없으므로, 이에 대한 처리를 한 후 수행을 종료하여야 한다.

Procedure *R3*
1. List update :
   Move *Ps* to the head of the appropriate list according to *VWs*
2. Release reservation value res :
   $RES_r = RES_r - res$.
3. $VW_r = K_r + RES_r - GUR_r$

(알고리즘 14) 예약값의 작업을 전송받지 못한 수신노드의 알고리즘

3.2.6 수신 노드 협상 시작 알고리즘

송신 노드 협상 시작 알고리즘과 마찬가지로, 분산 시스템의 각 노드들의 평균 부하가 정해진 임계값의 상한선 *upper_threshold*보다 더 큰 경우에는 송신 노드가 수신 노드를 찾을 확률보다 수신 노드가 송신 노드를 찾을 확률이 더 높아지며, 또한 이러한 경우 협상의 대상 노드를 찾는데 소요되는 메시지의 수도 현저하게 감소된다. 이 경우 대칭형 정책은 수신 노드가 대상 노드를 찾기 위한 협상을 시작하여야 한다.

수신 노드는 먼저 협상을 하기 위한 대상이 되는 잠정적인 송신 노드를 선택하여 대상 노드에게 자신의 보장값을 수신할 수 있는 지를 문의하는 폴링 메시지를 전송한다. 이 과정은 다음 (알고리즘 15)과 같다.

Procedure *R4*
1. Target node P_s is select :
   If $SList_r$ is not empty, choose the node at the head of $SList_r$.
   else, choose the last node from $RList_r$.
2. Determine reservation value *res* :
   res = *max* = *MaxAssign()*
3. Accumulate reservation value *res* :
   $RES_r = RES_r + res$
4. $VW_r = K_r + RES_r - GUR_r$
5. Send a polling message to P_s

(알고리즘 15) 수신 노드의 협상 시작 알고리즘

수신 노드 P_r로부터 폴링 메시지 *POLLING(res)*를 받은 송신 노드 P_s는 먼저 자신이 전송할 수 있는 작업의 크기를 결정한다. 이 과정에서 사용되는 함수가 *NumAssign()*이다. 일단 전송할 작업의 크기가 결정되면 자신의 작업 큐에서 작업을 선택하여 수신 노드 P_r로 작업을 전송하고, 전송후의 P_r의 부하 상태를 검사하여 리스트의 내용을 수정한다. 이 과정이 (알고리즘 16)에서 기술되어 있다.

Procedure S4
1. Determine transferring job size *b* :
   t = *NumAssign()*
   If $t \leq res$ then $b = t$
   else $b = res$
2. Select *b* tasks from task queue :
   If no task can be selected, goto procedure *S4'*
3. Transfer *b* tasks to P_r
   $K_s = K_s - b$
4. $VW_s = K_s + RES_s - GUR_s$
5. List update :
   Estimate P_r's new virtual load by calling *ReceiverNewVW()*.
   Move P_r to the head of the appropriate list of P_s accordingly

(알고리즘 16) 송신 노드의 작업 전송량 결정 알고리즘

협상에 참가한 송신 노드 P_s에서 전송되는 폴링 메시지의 형태는 *TRANSFER(tasks, b, res, VWs)*와 같다. 전송될 작업(*tasks*)과 작업의 크기(*b*), 예약값(*res*), 그리고 전송후의 송신 노드의 기대되는 가상 부하량(*VWs*) 등의 정보가 폴링 메시지와 함께 전달된다.

폴링 메시지를 받은 수신 노드 P_r은 이러한 정보를 근거로 리스트를 수정하고, 전송받은 작업을 자신의 임계 큐에 추가하며, 가상 부하값을 수정하고 알고리즘을 종료한다.

Procedure *S5*
1. List update :
   Move P_s to the head of the appropriate list according to VW_s
2. Append *b* tasks onto the threshold queue :
   $K_r = K_r + b$
3. Release reservation value res :
   $RES_r = RES_r - res$
4. $VW_r = K_r + RES_r - GUR_r$
5. Reset *probe_limit* counter and stop.

(알고리즘 17) 수신 노드로의 작업 전송 과정 알고리즘

프로시저 *S4*에서 전송할 작업의 크기가 결정되었으나 실제로 전송할 작업이 선택되지 못하는 경우가 발생할 수 있다. 이러한 경우에는 송신 노드의 가상 부하를 다시 검사하고, 이에 따라 적절한 조치를 하여야 한다. 이 때 수행하여야 할 알고리즘이 (알고리즘 18)에 기술되어 있다.

```
Procedure S4'
1. List update :
   Estimate P_r's new virtual load by calling ReceiverNewVW().
   Move P_r to the head of the appropriate list of P_s accordingly
2. Send a NACK message to P_r,
   maintain VW_s
```

(알고리즘 18) 송신 노드의 작업 선택이 불가능한 경우의 알고리즘

위의 *S4'* 프로시저에서 폴링 메시지 $NACK(res, VW_s)$를 전송하면 수신 노드 P_r은 현재의 잠정적인 송신 노드 P_s로부터 원하는 예약값만큼의 작업을 전송받을 수 없다. 따라서 은 폴링 메시지에 포함된 정보를 근거로 리스트의 내용을 수정한 후 다른 잠정적인 송신 노드의 선택을 하게 된다. 이 과정은 다음 (알고리즘 19)과 같다.

```
Procedure R6
1. Lists update :
   Move P_s to the head of the appropriate list according to VW_s
2. Release reservation value res :
   RES_r = RES_r - res
3. VW_r = K_r + RES_r - GUR_r
4. Initiate another polling session by going to procedure R4
   unless either
   a. probe_limit is exceeded or
   b. P_r is no longer a receiver (i.e. VW_r ≠ L-load) or
   c. No node deemed appropriate can be selected for polling.
```

(알고리즘 19) 다른 송신노드 선택 알고리즘

4. 시뮬레이션 모델

보장/예약 알고리즘의 성능은 시뮬레이션을 통하여 평가되었다. 한 노드 P_i에는 평균 도착률이 λ_i인 독립적인 Poisson 분포를 갖는다고 한다. 시스템에는 N개의 노드가 있다고 하고, N개의 노드($\{\lambda_1, \lambda_2, \lambda_3, \lambda_4, \cdots, \lambda_N\}$)의 평균 도착률은 평균이 λ_0이고, 표준 편차가 σ_λ인 정규형 지수분포라 가정한다. 이러한 분포는 다른 노드들이 서로 다른 작업 부하를 가질 때 사용된다. 이 분산의 표준 편차(σ_λ)는 분산 시스템의 부하 불균형의 양적 측정을 나타내고, 불균형 요소라 부른다. 한 작업이 처리되는데 필요한 CPU 시간을 작업 서비스 요구량이라 한다. 한 노드에서의 작업 서비스 요구량은 평균 S_0인 독립적인 지수 분포를 따른다고 한다. 이 시뮬레이션에서 S_0는 1로 가정한다[18].

4.1 비용 모델

기존의 많은 동적 부하 균등화 알고리즘들은 수행 오버헤드와 통신 오버헤드를 무시해도 좋을 만큼 가정을 지나치게 단순화하였다. 이러한 가정은 실행 시간의 정확한 상황을 반영하지 못하였다[9]. 동적 부하 균등화 알고리즘에서 정확한 성능과 효율을 측정하기 위해서 어느 정도의 CPU 오버헤드와 네트워크 지연의 모델링은 필수적이라 할 수 있다. Eager et al.의 논문에서 규정한 CPU 오버헤드는 작업 서비스 요구의 일부분으로서 규정하였다[4, 13]. 그러나 이 논문들은 폴링과 관련된 CPU 오버헤드를 반영하지는 못하였다. 본 논문에서 사용하는 모델에서는 폴링의 처리와 관련된 CPU 오버헤드를 고려하였다.

통신상의 지연은 분산 시스템의 특징이고, 동적 부하 균등화 알고리즘의 부하 정보와 관련된 정책의 설계에 있어서는 중요한 요소이다. 이것은 부하 상태 정보의 정확성에 있어서 실제적인 효과를 미치고, 역으로 스케줄링의 결정에 큰 영향을 미친다. 본 모델에서는 모든 메시지 교환에서의 통신 지연을 계산하였다.

4.1.1 폴링 질의 비용 모델

작업 전송의 대상자를 정함에 있어 잠정적인 전송 대상자로부터 동의를 얻거나 부하 상태를 조사하기 위하여 질의 메시지의 전송이 필요하다. 본 논문에서는 폴링 메시지의 처리를 위한 CPU 오버헤드를 송수신 노드 모두 동일하다고 가정한다. 이 CPU 오버헤드를 $CPU_{overhead}$라 한다.

그리고 폴링 메시지의 전송을 위한 통신 지연 또한 무시할 수 없는 것이다. 폴링 메시지는 속성상 길이가 매우 짧기 때문에 본 논문에서는 단지 하나의 메시지 송출 비용 $F_{polling}$이 필요하다고 가정한다. 그러므로 하나의 폴링 메시지를 전송할 때의 통신 지연 $DELAY_{polling}$은 다음과 같이 정의할 수 있다.

$$DELAYpolling = Fpolling + D$$

여기서 D는 네트워크 전송시의 지연이다.

4.1.2 작업 할당 비용 모델

작업의 할당은 한 호스트에서 다른 호스트로의 작업의 전송과 무관하지 않다. 작업 할당에서의 CPU 오버헤드는 폴링 메시지 처리비용보다는 훨씬 더 크다. 본 논문에서는 송신 노드와 수신 노드의 CPU 오버헤드가 동일하다고 가정한다. 전송 작업의 크기는 작업 전송시 CPU 오버헤드에 결정적인 영향을 미친다. 전송 작업의 크기가 커질수록 CPU 오버헤드가 커진다. 작업 전송시 CPU 오버헤드는 다음과 같이 정의한다.

정의 1 : 작업 할당과 관련된 CPU 비용 $CPU_{assignment}$는 다음과 같다.

$$CPU_{assignment} = C_{assign} + C_{pack} * \sum_{i=1}^{b} l_i \qquad (8)$$

여기서 Cassign은 할당 알고리즘을 실행하는데 필요한 CPU 시간이고, Cpack는 각 작업 메시지를 구성하고 해제하는데 필요한 CPU 시간이다. b는 전송할 작업의 크기이고, li는 작업 i가 생성하는 메시지의 수이다. 따라서 li는 작업 i의 코드 길이라 한다. 따라서 $\sum_{i=1}^{b} l_i$는 작업군 이생성하는 전체 메시지의 수를 나타낸다. 한 노드에서 li는 평균이 lo인 독립 지수 분포를 가진다.
하나의 작업군을 전송하는데 필요한 통신 지연은 폴링 처리비용보다 현저히 크다.

정의 2 : 일련의 작업군을 전송할 때의 통신 지연을 다음과 같이 정의한다.

$$DELAY_{assignment} = (F_{task} * \sum_{i=1}^{b} l_i) + D \qquad (9)$$

여기서 F_{task}는 하나의 작업 전송 메시지를 통신 채널로 전송할 때 필요한 시간이다.

4.1.3 실행 우선순위

부하 균등화와 관련된 작업(예 : 폴링, 할당 등)의 우선순위는 일반적인 사용자 작업보다 우선순위가 높다. 부하 균등화 알고리즘이 호출되면 수행중인 사용자 작업은 일시 정지된다. 부하 균등화와 관련된 수행은 FIFO 형태로 스케줄링이 수행된다. 부하 균등화와 관련된 모든 수행이 종료될 때까지 사용자 작업은 재시작되지 않는다.

4.1.4 성능 측정 요소

〈표 2〉 성능 측정 요소

성능 측정 요소	의 미
평균 작업 응답 시간	부하 균등화 알고리즘의 성능을 측정하는 주 요소
CPU 오버헤드의 비율	부하균등화 알고리즘을 실행하는데 소요된 총 CPU시간. 성능 개선을 얻기 위하여 부하 균등화 알고리즘을 실행함으로써 발생하는 CPU 오버헤드의 수준을 측정한다.
전송 작업 크기	작업군에 포함된 평균 작업 수

위의 <표 2>는 부하 균등화 알고리즘을 평가할 때 사용되는 성능과 효율에 관련된 일반적인 요소이다.

다른 부하 균등화 알고리즘과 본 논문에서 제안한 알고리즘을 프로세서 쓰레싱의 수준에서 비교하기 위하여 본 논문에서는 프로세서 쓰레싱의 양적 측정을 위하여 쓰레싱 공동 계수라 불리는 새로운 행렬을 제시한다. 프로세서 쓰레싱은 송신 노드 쓰레싱과 수신 노드 쓰레싱 두가지 형태가 있다. 송신노드 쓰레싱은 잠정적인 송신 노드가 많은 수의 수신 노드에 의하여 폴링이 될 때를 일컫는다. 수신 노드 쓰레싱이란 잠정적인 수신 노드가 많은 송신 노드에 의해 폴링이 될 때를 말한다. 본 논문에서는 송신 노드의 쓰레싱 공동 계수 V_s를 다음과 같이 정의한다.

$$V_s = \frac{\sqrt{\frac{\sum_{i=1}^{N}(S')^2 - N(\overline{S'})^2}{N}}}{\overline{S'}} = \frac{\sigma S'}{\overline{S'}}$$

여기서 S'_i는 노드 P_i가 수신 노드로부터 폴링되는 총 횟수이고, $\overline{S'}$은 S'_i 의 평균으로 $\sum_{i=1}^{N} S'_i / N$ 이다. $\sigma S'$은 변수 S'_i의 표준 분산이다.
이와 마찬가지로 수신 노드의 쓰레싱 공동 계수 V_r을 다음과 같이 정의한다.

$$V_r = \frac{\sqrt{\frac{\sum_{i=1}^{N}(R')^2 - N(\overline{R'})^2}{N}}}{\overline{R'}} = \frac{\sigma R'}{\overline{R'}}$$

V_s와 V_r은 각각의 폴링으로 인한 분산의 정도를 측정한다. V_s(혹은 V_r)의 값이 크다는 것은 모든 송신(수신) 노드가 폴링을 여러 수신(송신)노드로부터 받았다는 것을 의미한다. 이것은 송신 노드(수신 노드)의 쓰레싱의 정도가 크다는 것을 의미한다.

5. 시뮬레이션 결과 및 분석

이 장에서는 본 논문에서 제안한 보장/예약 기법의 성능을 검증하기 위하여 시뮬레이션한 결과를 분석하였다. 시뮬레이션을 위하여 사용된 도구는 K. T. S. Co., Ltd의 Siman IV이며 실험 환경은 시스템은 Pentium III 400MHz, Windows 98이다.

5.1 시뮬레이션 매개변수

본 시뮬레이션에서 사용된 매개변수의 값은 아래의 <표 3>과 같다. 이 매개변수의 값은 본 논문의 근간이 되는 SK알고리즘에서 사용된 값과 동일한 범주이다[10].

〈표 3〉 시뮬레이션 매개변수 값

매 개 변 수	값
N	30
T_{CPU}	0.1
σ_A	3
S_o	1
lower_threshold	5
upper_threshold	20
probe_limit	5
$F_{polling}$	0.001
F_{task}	0.005
D	0.005
C_{pack}	0.003
C_{assign}	0.002
l_o	5

앞에서 설명한 바와 같이 N은 시스템을 구성하는 노드의 수이다. T_{CPU}는 각 작업들에게 CPU가 할당되는 단위 즉,

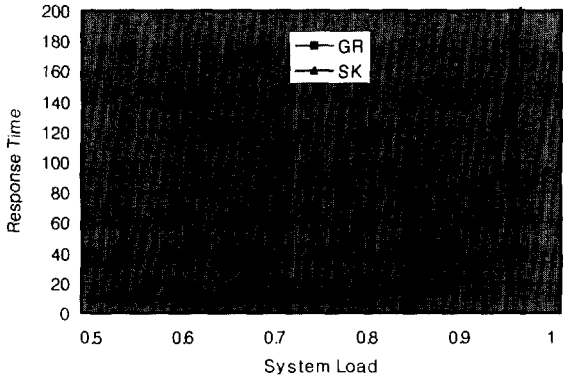
time slice를 의미하며, σ_λ는 작업 도착률의 표준편차이다. S_o는 작업 서비스 요구량의 평균이고, *lower_threshold*와 *upper_threshold*는 시스템이 저부하 상태와 과부하 상태가 되는 임계값이다. *probe_limit*는 송신 노드 혹은 수신 노드가 작업의 전송을 위하여 협상을 시도하는 횟수를 나타내며, $F_{polling}$과 F_{task}는 하나의 폴링 메시지와 하나의 작업을 전송할 때의 비용이다. D는 네트워크 전송시의 지연을 의미하며, C_{pack}는 각 작업 메시지를 구성/해체하는데 필요한 시간이며, C_{assign}은 할당 알고리즘을 실행하는데 필요한 시간이다. 그리고 l_o는 하나의 작업이 생성하는 전체 메시지의 수의 평균으로 지수 분포를 따른다.

5.2 시뮬레이션 결과

본 논문에서 제안한 보장/예약 알고리즘과 대칭형 위치 정책을 제안한 Shivaratri와 Krueger의 단일 작업 할당 알고리즘의 성능을 시뮬레이션을 통하여 비교하였다. 편의를 위하여 본 논문에서 제안한 보장/예약 알고리즘을 GR이라 하고, Shivaratri와 Krueger가 제안한 알고리즘을 SK라 한다.

5.2.1 평균 응답 시간

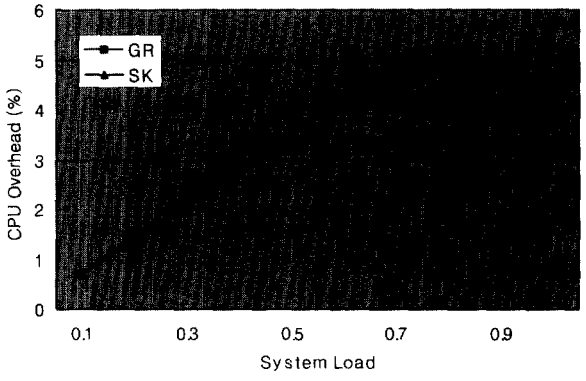
(그림 2)에서와 같이 시스템의 평균 작업이 약 60%이하에서는 두 개의 알고리즘의 평균 응답 시간이 동일하다. 즉 보장/예약 알고리즘의 전송 크기가 시스템의 부하가 낮은 경우에는 SK의 전송 크기와 동일하다는 것이다. 그러나 시스템의 부하가 약 85%에 도달하게 되면 SK 알고리즘은 포화상태가 되어 시스템의 성능이 급격하게 떨어진다. 이에 비하여 GR 알고리즘은 SK에 비해 더 좋은 성능을 제공한다. 이러한 사실로 미루어 볼 때, GR 알고리즘은 SK에 비해 전체 시스템의 부하가 커질수록 부하의 분산이 적절하다는 것을 의미한다.



(그림 2) 평균 응답 시간

5.2.2 CPU 오버헤드

하나의 작업이 전송될 때 GR 정책은 SK 정책에 비해 더 많은 메시지의 교환을 수행한다. 시스템의 부하가 약 60%부근까지는 SK에 비해 GR이 오버헤드가 더 크게 나타난다. 그러나 시스템의 전체 부하가 낮을 경우에는 이러한 오버헤드를 처리할 만큼의 충분한 여유가 있기 때문에 (그림 3)에서와 같이 심각한 성능 저하가 일어나지 않는다. 그러나 시스템의 부하가 높은 경우에는 조그만 오버헤드도 시스템의 성능에 지대한 영향을 미칠 수 있다. 그리고 GR 정책과 비교해볼 때 SK 정책이 동일한 작업을 전송하는데 소요되는 폴링의 수가 더 많으며, 또한 작업 전송을 위한 협상 대상 노드를 찾는데 실패할 확률이 더 낮다



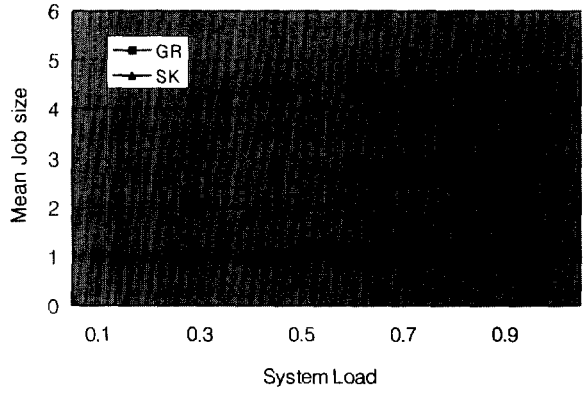
(그림 3) CPU 오버헤드

5.2.3 전송 작업의 크기

SK 정책은 한번의 폴링으로 하나의 작업만을 전송할 수 있도록 설계되어 있다. 따라서 이 정책은 시스템의 부하에 무관하게 전송할 작업의 크기가 항상 1이다.

그러나 GR 정책은 시스템의 부하가 60% 이하일 경우에는 SK 정책의 전송 작업의 크기와 동일하다.

이것은 시스템의 부하가 낮을 경우에는 각 노드의 작업 부하의 차이가 심하지 않아 전송할 작업이 많지 않음을 의미한다. 그러나 시스템의 부하가 높아질수록 작업 부하의 차이가 점차 커지게 되므로 전송할 작업의 크기 또한 커지는 것이 당연할 것이다. 무엇보다 전송 작업의 크기를 결정하는 요소는 작업 도착률의 표준편차 σ_λ이다. 본 논문에서는 이 값을 3으로 하여 실험하였다.



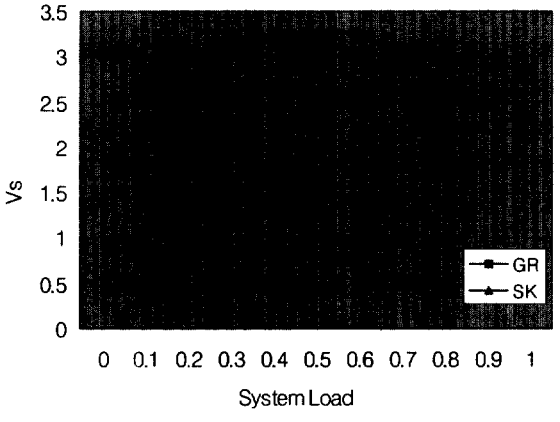
(그림 4) 전송 작업의 크기

5.2.4 수신 노드 쓰레싱 공통계수(Vr)

수신 노드 쓰레싱이란 잠정적인 수신 노드가 많은 송신 노드에 의해 폴링이 될 때를 말한다. 송신 노드의 쓰레싱 공동 계수는 송신 노드의 폴링수의 분산을 평균으로 나눈 값이다. 여기서 공통 계수의 값이 크다는 것은 수신 노드가 폴링을 여러 송신 노드로부터 받았다는 것을 의미한다. 즉, 수신 노드의 쓰레싱의 정도가 크다는 것을 의미한다.

시스템의 부하가 낮은 경우에는 대부분의 노드가 저부하 상태가 되어 잠정적으로 수신 노드가 되고, 단지 일부 노드만이 송신 노드가 될 것이다. SK 정책의 경우, 특정 송신 노드가 수신 노드로부터 협상을 받을 확률은 매우 높다. 그러므로 시스템의 부하가 낮은 경우에는 송신 노드가 시작하는 폴링의 대상이 되는 수신 노드는 그 수가 적다. 바꾸어 말하면 이것은 저부하 시스템에서는 수신 노드의 쓰레싱이 높다는 것이다.

그러나 GR 정책의 경우에는 송신 노드는 현재 협상중인 노드에게 일련의 작업을 전송한 후에야 비로소 다른 노드와 협상을 시작한다. 이것은 현재의 수신 노드의 여분의 처리 능력을 모두 사용하게 되므로 SK에 비해 쓰레싱이 줄어든다는 것을 의미한다. 다음의 (그림 5)에서 이러한 결과를 보이고 있다.

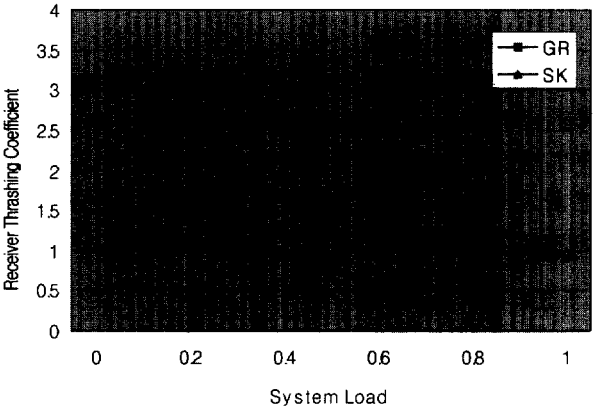


(그림 5) 송신 노드 쓰레싱 공통 계수(Vs)

5.2.5 송신 노드 쓰레싱 공통계수(Vs)

5.2.4의 수신 노드 쓰레싱 공통 계수와 마찬가지로 송신 노드 쓰레싱이란 잠정적인 송신 노드가 많은 수의 수신 노드에 의해 폴링이 될 때를 말한다. 여기서 V_s의 값이 크다는 것은 송신 노드가 폴링을 여러 수신 노드로부터 받았다는 것을 의미한다. 즉, 송신 노드의 쓰레싱의 정도가 크다는 것을 의미한다.

두 정책 모두 시스템의 부하가 약 40%이하일 경우, 시스템의 부하가 증가할수록 송신 노드의 쓰레싱이 증가한다. SK 정책에서 수신 노드가 협상을 시작할 경우, 대부분의 노드들이 수신 노드 상태이므로 시스템에는 많은 수의 폴링 메시지가 발생하지만, 송신 노드는 상대적으로 그 수가 매우 적다. 따라서 각 송신 노드의 쓰레싱 공통 계수의 값은 커지며, 이로 인하여 상대적인 처리 속도의 저하를 가져온다. 그러나 시스템의 부하가 약 50% 이상이 되면, 수신 노드의 수가 감소하며, 반대로 송신 노드의 수가 증가하게 되므로 쓰레싱 공통 계수의 값이 감소하게 된다. (그림 6)에서 송신 노드의 쓰레싱 공통계수의 값의 변화를 기술하고 있다.



(그림 6) 수신 노드 쓰레싱 공통계수

6. 결 론

동적 부하 분산 정책은 정책의 시작 위치에 따라 송신 노드 시작 정책과 수신 노드 시작 정책, 그리고 송신 노드와 수신 노드 모두에서 시작하는 대칭형 정책으로 나뉜다. 송신 노드 시작 정책은 전송할 작업을 가진 과부하 노드가 부하 분산을 시작하는 방법으로 시스템의 평균 부하가 낮은 경우에 좋은 성능을 발휘하며, 반대로 수신 노드 시작 정책은 저부하 노드가 과부하 노드를 찾는 방법으로 시스템의 평균 부하가 높을 경우에 유리하다.

이 두가지 정책의 장점만을 고려하여 시스템이 저부하인 경우에는 송신 노드 시작 정책을, 그리고 과부하인 경우에는 수신 노드 시작 정책을 사용하는 것이 대칭형 정책이다. 그러나 대칭형 정책은 프로세서 쓰레싱 현상이 발생할 수 있다.

프로세서 쓰레싱이란 여러개의 노드가 동시에 부하의 분산을 위하여 동일한 노드에게 협상을 시도하는 상황을 말한다. 프로세서 쓰레싱 상황이 CPU로 하여금 과도한 메시지의 처리를 요구하여 동적 부하분산 정책의 성능이 저하될 수 있다.

따라서 본 논문에서는 부하 분산시의 프로세서 쓰레싱의

빈도를 감소시키기에 적합한 알고리즘을 제안하였다. 이 알고리즘은 다음의 세가지 요소로 구성되어 있다. 첫 번째는 송신 노드가 수신 노드에게 전송할 작업의 양을 결정하는 알고리즘이다. 이 알고리즘에서는 한번의 작업 이주시 하나의 작업을 전송하는 기존의 대칭형 알고리즘과 달리, 송신 노드와 수신 노드의 부하차이를 연산하여 한번의 작업 이주로 부하 균등화를 이룰 수 있다.

두 번째는 송신 노드와 수신 노드 사이에서 전송할 작업의 크기에 관하여 상호 동의를 구하는 협상 프로토콜이다. 협상 대상이 선택되면 송신 노드가 전송할 작업량을 보장하고, 수신 노드가 전송받을 수 있는 작업량을 미리 예약한다. 이렇게 함으로써 송수신 노드 모두 한번의 협상이 시작되면 다른 노드의 협상 대상에서 제외되어 쓰레싱 현상을 방지할 수 있다.

그리고 세 번째는 잠정적인 전송 대상을 선택하기 위한 대칭형 위치 정책의 설계이다. 시스템의 임계 큐에 존재하는 작업의 크기에 따라 송신 노드 시작 정책과 수신 노드 시작 정책을 동적으로 수행함으로써 대칭형 정책을 수행할 수 있다. 그리고 이주된 작업을 임계 큐에 위치시킴으로써 이주된 작업이 다시 재배치되는 현상을 방지한 시스템 모델도 고안되었다. 또한 이 알고리즘은 이기종 분산 시스템에서의 부하 분산에도 적용하기 위한 방안도 고려되었다. 노드들의 속도차이를 전송량에 적용하기 위하여 작업 부하의 정량화 방안을 고안하였다.

본 논문에서 제안한 알고리즘의 성능 개선을 증명하기 위하여 Siman IV 시뮬레이션 도구를 이용한 모의실험을 통하여 알고리즘의 성능을 분석하고 제안된 알고리즘을 기존 대칭형 정책과 비교하여 평가하였다.

평가 요소로는 평균 응답시간과 CPU 오버헤드, 그리고 송수신 노드의 쓰레싱 정도를 나타내는 쓰레싱 공통 계수를 측정하였다.

평균 응답 시간은 시스템이 저부하일 경우에는 기존 방법과 큰 차이를 보이지 않으나 평균 응답 시간이 급격히 증가하는 시스템 과부하일 경우에는 기존 연구에 비해 현저한 성능 증가를 보였다.

CPU 오버헤드의 비율 또한 시스템의 과부하 상태에서는 본 논문에서 제안한 알고리즘이 기존의 알고리즘에 비해 더 낮다.

쓰레싱 공통계수는 송신 노드와 수신 노드에 따라 그 값이 다르게 나타난다. 본 논문에서 제안한 알고리즘이 수신 노드 쓰레싱 공통계수의 경우 시스템의 부하가 낮을 때 발생 빈도가 낮으며, 송신 노드 쓰레싱 공통계수는 시스템의 부하가 높을 때 더 유리하다. 따라서 시스템의 임계 큐의 상한값과 하한값을 적절하게 선택한다면 모든 시스템 상황에서 쓰레싱의 발생 빈도를 낮출 수 있다.

참 고 문 헌

[1] Marvin M. Theimer and Keith A. Lantz. "Finding Idle Machine in a Workstation-Based Distributed systems," IEEE Transactions on Software Engineering, Vol.15, No.11, November 1989.

[2] L. M. Ni, C. W. Xu, and T. B. Gendreau. "Load Balancing form a UNIX Shell," In Proceedings, the 13th Conference on Local Computer Networks, October 1988.

[3] Matt Bishop, Mark Valence, and Leonard F. Wisniewski. "Process Migration for Heterogeneous Distributed Systems," Technical Report TR95-264, Dartmouth College, Aug. 1995.

[4] D. L. Eager and E.D. Lazowska. "Adaptive Load Sharing in Homogeneous Disributed Systems," IEEE Transactions on Software Engineering, Vol.SE-12, No.5, May 1986.

[5] C. Jacqmot and E. Milgrom. "A Systematic Approach to Load Distribution Strategies for Distirbuted Systems," In Decentralized and Distributed Systems, Sep. 1993.

[6] Andrew Murray Bond. "Adaptive Task Allocation in a Distributed Workstation Environment," PhD. thesis Victoria University of Wellington, 1993.

[7] S. Zhou. "A Trace-Driven Simulation Study of Dynamic Load Balancing," IEEE Transactions on Software Engineering, Vol.14, No.9, pp.1327-1341, Sep. 1988.

[8] Ishfaq Ahmad, Arif Ghafoor, and Kishan Mehrotra. "Performance Prediction of Distributed Load Balancing on Multicomputer Systems," In Proceedings, Supercomputing '91, pp.830-839, 1991.

[9] O. Kremien and J. Kramer. "Methodical Analysis of Adaptive Load Sharing Algorithms," IEEE Transactions on Parallel and Distributed Systems, Vol.3, No.6, pp.747-760, Nov. 1992.

[10] N. G. Shivaratri and P. Krueger. "Two Adaptive Location Policies for Global Scheduling Algorithms," In Proceedings, The 10th International Conference on Distributed Computing Systems, pp.502-509, May 1990.

[11] Chin Lu and Sau-Ming Lau. "A Performance Study on Load Balancing Algorrithms with Process Migration," In Proceedings, IEEE TENCON 1994, pp.357-364, Singarpore, Aug. 1994.

[12] Thomas Kunz. "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme," Technical Report TI-6/91, Institute for Theoretische Informatik, Fachbereich Informatik, Technische Hochschule Darmstadt, Dec. 1991.

[13] S. Zhou, X. Zheng, J. Wang, and P. Delisle, "Utopia : a Load Sharing Facility for Large, Heterogeneous Distribut-

ed Computer Systems," Software-Practice and Experience, Vol.23, No.12, pp.1305-1336, Dec. 1993.

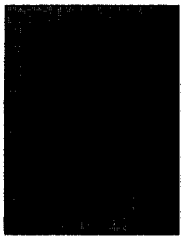
[14] N. G. Shivaratri, P. Krueger, and M. Singhan, "Load Distributing for Locally Distributed Systems," IEEE Computer, pp.33-44, Dec. 1992.

[15] K. Benmohammed-Mahieddine and P. M. Dew. "A Periodic Symmetrically-Initiated Load Balancing Algorithm for Distributed Systems," SIGOPS, Vol.28, No.1, pp.66-77, Jan. 1994.

[16] Chin L. U. and Sau-Ming L. A. U. "An Adaptive Load Balancing Algorithm for Heterogeneous Distributed Systems with Multiple task classes," Proceedings of 16th Distributed Computing Systems, May. 1996.

[17] Jun-Yeon Lee, Young-Chan Kim. "Synchronization algorithm of migrating service object," Proceedings of the High Performance Computing Conference, pp.1407-1413, Sep. 1998.

[18] 이준연, 김대현, 김영찬. "클라이언트/서버 응용의 연산 부하 측정을 위한 시뮬레이터", 정보과학회논문지(C), Vol.5, No.2, pp.185-195, 1999.



이 준 연

e-mail : jylee@tmic.tit.ac.kr

1990년 중앙대학교 전산과 졸업(학사)

1992년 중앙대학교 대학원 전산과 졸업(석사)

1992년~1994년 (주)Microsoft

2000년 중앙대학교 대학원 전산과 졸업(박사)

2000년~현재 동명정보대학교 멀티미디어공학과 전임강사

임 재 현

e-mail : defacto@intizen.com

1986년 중앙대학교 전자계산학과 졸업(이학사)

1988년 중앙대학교 일반대학원 전자계산학과(이학석사)

1998년 중앙대학교 일반대학원 컴퓨터공학과(공학박사)

1998년~2000년 공주문화대학 컴퓨터정보과 전임강사

2001년~현재 공주대학교 영상정보공학부 조교수

관심분야 : 클라이언트/서버 시스템, 정보검색, 분산운영체제