프로세스 재순서에 기반한 동시 스케쥴링 기법

유정록† 드리스† 김진수‡ 맹승렬†
† {jlyu, driss, maeng}@calab.kaist.ac.kr
‡ jinsoo@cs.kaist.ac.kr

PROC : Process ReOrdering-based Coscheduling on Non-dedicated NOWs

Jung-Lok Yu, Driss Azougagh, Jin-Soo Kim, Seoung-Ryoul Maeng
Division of Computer Science, Korea Advanced Institute of Science and Technology

요 약

최근의 Network Of Workstations (NOW) 시스템은 범용의 컴퓨팅 플랫폼으로 각광을 받고 있지만, 그 활용률은 여전히 사용자의 기대에 미치지 못하고 있다. 본 논문에서는 NOW 시스템의 활용률을 높이기 위한 방법으로 시스템의 부하 수준을 고려하여 프로세스 재순서 (process reordering)를 사용하는 동시 스케쥴링 기법인 PROC (Process ReOrdering-based Coscheduling)을 제안한다. 그리고 자세한 시뮬레이션을 통해 PROC의 우수성을 검증한다.

1. 서 론

최근들어, 상용의 컴퓨팅 자원들을 고속의 네트워크로 연결한 Network of Workstations (NOW)이 범용의 컴퓨팅 플랫폼 (과학 계산용 서버, 멀티미디어, 데이터베이스, 웹 서버 등)으로 각광을 받고 있다. NOW는 가격 대비 성능비, 고가용성, 확장성등의 장점을 가지고 있지만, 그 활용률 (utilization)은 여전히 사용자의 기대에 미치지 못하고 있다.

NOW의 활용률을 제한하는 요소는 크게 세 가지로 나누어 볼 수 있다: 1) 컴퓨팅 자원간의 통신 비용, 2) 동기화가 필요한 프로세스들의 스케쥴링, 3) 부하 불균형이 바로 그것이다.

첫번째 한계점을 해결하기 위해서 산업, 학계에서는 기존의 TCP/IP 통신 프로토콜에서 반드시 필요한 시스템 콜, 메모리 복사, 문맥 교환 (context switching)의 오버헤드를 극복하고 최소 지연시간, 최대 대역폭을 가능케 하는 System Area Network (SAN)과 사용자 수준 통신 프로토콜를 제안하였다. 대표적인 예로 VMMC, U-Net, FM, AM, BIP, VIA등이 있다.

그러나 이러한 통신 성능의 비약적인 발전만으로는 NOW의 활용률을 높이기 힘들다. 왜냐하면 메시지를 주고 받은 프로세스들을 스케쥴링하는 로컬 스케쥴러들은 독립적으로 동작하기 때문에 병렬 프로세스들간의 동기화 시간이 증가할 수 있기 때문이다. 이러한 비효율적인 스케쥴링 방법들을 극복하기 위해서 최근에 다양한 동시 스케쥴링 (Coscheduling) 기법들이 제안되었다. 동시 스케쥴링의 기본 아이디어는 병렬 프로세스간에 내부적으로 발생하는 통신 이벤트를 동시 스케쥴링의 트리거 (trigger)로 사용하는 것이다. 제안된 기법들에는 Demand-based CoScheduling (DCS), Spin-Blocking (SB), Periodic Boost (PB)등이 있다 [1].

마지막으로 부하 불균형은 하드웨어적인 이질성, 동질의 하드웨어상에서 서로 다른 부하 요소 (computation, communication, I/O 비율)를 갖는 프로세스들의 수행 등으로 인해서 나타난다. 이러한 부하 불균형은 동기화가 필요한 프로세스들의 바쁜 대기 (busy waiting) 및 유휴시간의 증가를 야기시키기 때문에 NOW의 활용률을 추가적으로 제한하는 요소이다 [2].

그러나 기존에 제안된 동시 스케쥴링 기법들에서는 이러한 부하 불균형을 전혀 고려하고 있지 않다. 따라서 본 논문에서는 시스템 부하 수준을 고려하여 프로세스의 재순서 (process reordering)를 사용하는 동시 스케쥴링 기법 (Process ReOrdering-based Coscheduling: PROC)을 제안한다. 또한 자세한 시뮬레이션을 통해 PROC의 우수성을 검증한다.

논문의 구성은 다음과 같다. 2절에서는 기존의 동시 스케쥴링 기법들에 대해서 간략히 살펴보고, 3절에서 PROC 스케쥴링 기법을 자세히 기술한다. 4절에서는 시뮬레이션 방법 및 실험 결과에 대해서 기술한다. 마지막으로 5절에서는 결론을 맺는다.

2. 동시 스케쥴링 기법

동시 스케쥴링 기법은 크게 두 가지 (메시지를 기다리는 행동, 메시지 수신시 행동)의 요소를 통해서 표 1과 같이 분류될 수 있다.

	Busy Wait	Block	Spin Block	Spin Yield
None	SPIN	IB	SB	SY
Interrupt and Boost	DCS	IB-DCS	SB-DCS	SY-DCS
Periodically Boost	PB	IB-PB	SB-PB	SY-PB

표 1. 동시 스케쥴링 기법들

SPIN은 가장 기본적인 방법으로 프로세스가 Busy Waiting을 수행할 때 메시지가 수신되어야 동시 스케쥴링이 된다. IB (Immediate Blocking)에서는 메시지 수신 과정에서 메시지가 아직 도착하지 않았으면 그 프로세스는 바로 블록된다. SB (Spin Blocking) 기법은 SPIN과 IB의 중간적인 기법으로 메시지

수신시 미리 정의된 고정 시간 동안 spinning을 수행하고, 그 시간 전에 메시지가 수신되지 않으면 블록된다. 이후에 메시지가 도착하면 인터럽트 서비스 루틴 (ISR)을 통해서 그 프로세스는 다시 깨어나게 된다. DCS (Demand-based CoScheduling)에서는 메시지를 수신하는 과정에서 프로세스는 Busy Waiting을 수행한다. 그리고 네트워크 인터페이스 (NI)에 메시지가 도착했을 때, 메시지의 목적 프로세스가 현재 스케쥴링이 되어 있지 않다면 NI는 인터럽트를 발생시키고, ISR에서는 목적 프로세스의 등급 (priority)을 올리게 된다. PB (Periodic Boost)는 DCS, SB에서의 인터럽트 비용을 제거하기 위해서 제안된 방법으로 하나의 커널 쓰레드가 프로세스들의 메시지 큐를 주기적으로 조사하고, 큐에 매달린 메시지가 있을 때 프로세스의 등급 조정을 통해서 동시 스케쥴링을 수행한다.

3. Process ReOrdering based Coscheduling (PROC)

그림 1은 부하 불균형이 야기하는 문제점을 기술한 것이다. 그림 1(1)에서 보듯이, N_2에서 동시 스케쥴링의 대상이 되는 프로세스가 여러 개 존재할 때, 부하 불균형을 고려하지 않으면 원격 노드에서의 SPIN 시간의 증가를 가져온다. 이렇게 부하 불균형이 전체 NOW의 활용률을 제약하는 요소임에도 불구하고 기존의 동시 스케쥴링 기법들은 이를 전혀 고려하고 있지 않다. 본 연구에서 제안한 PROC에서는 전체 시스템의 활용률을 높이기 위해서 동적인 부하 수준에 대한 런타임 측정을 수행하고, 이렇게 얻어진 부하 정보를 기존의 통신 메시지에 실어서 (piggybacking) 다른 노드들로 전파한다. 그리고 원격 노드들의 부하 정보를 받은 로컬 노드에서는 그 부하 정보들을 기반으로 프로세스 재순서 (그림 1(2) 참조)를 통해서 효율적인 동시 스케쥴링을 수행한다.

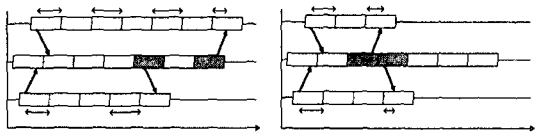

그림 1. 부하 불균형

제안한 PROC 기법에서 독립적인 노드들이 계산하는 부하 정보는 크게 두 가지로 이루어져 있다. 첫째는 노드 *i*에서 프로세스들이 평균적으로 소비하는 Time Slice (TS_i) 이고, 둘째는 현재 스케쥴링된 프로세스가 향후에 재스케쥴링 될때까지 노드 *i*에서 실행될 예상 프로세스 개수 (ENP_i)이다. ENP_i는 다시 노드 *i*에서 매달린 메시지 (pending message)를 갖는 프로세스의 개수, 노드 *i*에서 highest-level ready 큐에 있는 프로세스의 개수, 노드 *i*에서 특정 기간 동안 메시지의 도착으로 인해서 깨어나는 (wake-up) 평균적인 프로세스의 개수를 통해 계산된다.

N개의 노드로 이루어진 시스템에서 *i* 번째 노드 N_i가 자신의 부하 정보인 (TS_i, ENP_i)를 노드 N_j의 프로세스 P_k에게 전송했을 때, 우리는 다음을 정의한다:

- TS_{ijk} 와 ENP_{ijk} : $TS_{ijk} \leftarrow TS_i$, $ENP_{ijk} \leftarrow ENP_i$
- T_{ijk} : N_j의 P_k가 N_i로부터 마지막으로 메시지를 받은 시간
- T_{ij} : N_j가 N_i로부터 마지막으로 메시지를 받은 시간
- TS_{ij} : N_j가 N_i로부터 받은 최신의 TS_i

N_j가 N_i로부터 새로운 메시지를 받았을 때, N_j의 NIC는 위의 네 가지 정보를 스케쥴링 계층의 자료 구조에 업데이트를 수행한다. 따라서 메달린 메시지를 가진 프로세스들은 메시지의 송신 노드들 (원격 노드들)의 최신 부하정보를 보유하게 되고, 스케쥴러는 이러한 부하정보들을 기반으로 프로세스들의 수행 순서를 결정하고 가장 긴급한 프로세스를 먼저 실행하게 된다. 그림 2는 PROC에서의 프로세스 재순서 예제를 보인 것이다. (N_j에서는 N_h와 N_i의 부하 정보인 ENP, TS를 기반으로 Expected Remaining Time (ERT)을 계산하게 되고, 계산된 ERT를 기준으로 프로세스 재순서를 수행한다.)

그림 2. 프로세스 재순서 예제

PROC의 프로세스 재순서 알고리즘은 아래 그림 3과 같다. 먼저 시간의 흐름에 따라 원격 노드들의 부하 정보를 업데이트하고, 메달린 메시지를 갖는 프로세스에서 최소의 ERT_{ijk} 값을 계산한다. 그리고 여러 개의 프로세스들 중 최소의 ERT_{ijk} 값을 가진 프로세스를 Candidate Set of Preferable process (CSP) 모임에 넣게 된다.

```
1) Reordering Procedure (node N_j, current time t, CSP)
2) {
3)     CSP = null;
4)     ERF_j = infinite;
5)     for each process P_k with pending message(s) in N_j
6)     {
7)         ERT_jk = infinite;
8)         for each message m of P_k
9)         {
10)            i = sender node of message m;
11)            /* maintain the load imbalance information */
12)            if( T_ijk < T_ij )
13)            {
14)                ENP_ijk = ENP_ijk - ((T_ij - T_ijk) / TS_ij);
15)                T_ijk = T_ij;
16)            }
17)            ERT_jk = (TS_ij * ENP_ijk) - (t - T_ijk);
18)            /* determine minimum ERT value in a process */
19)            if( ERT_jk > ERT_ijk ) ERT_jk = ERT_ijk;
20)        }
21)        /* determine a process set with minimum ERF value */
22)        if( ERF_j > ERT_jk ) { ERF_j = ERT_jk; CSP = {p}; }
23)        else if( ERF_j == ERT_jk ) CSP = CSP + {p};
24)    }
25) }
```

그림 3. 프로세스 재순서 알고리즘

본 연구에서는 PROC의 성능을 비교하기 위한 비교 대상으로 가장 대표적인 동시 스케쥴링 기법은 SB와 PB를 사용하였다.

4. 성능 비교

4.1 시뮬레이터 및 실험 방법

PROC과 기존의 동시 스케쥴링 기법들과의 성능 비교를 위해서 본 연구에서는 CSIM19 [3] 시뮬레이션 툴킷을 사용하여 정밀한 process-oriented, event-driven 시뮬레이터를 제작하였다. 시뮬레이션 모델에서 각 노드들은 네트워크 인터페이스, OS 스케쥴러, 어플리케이션 프로세스들로 구성된다. 그리고 이러한 컴퓨팅 자원들을 묶는 네트워크 모듈이 있고, 네트워크 모듈에서는 경쟁이 없는 간단한 선형 모델을 사용하였다. OS 스케쥴러는 솔라리스 스케쥴러를 모델링하였고, DCS, SB 기법을 모델링하기 위한 인터럽트 서비스 루틴 모듈과 PB 기법을 모델

링하기 위한 커널 쓰레드 모듈을 각 노드에 추가하였다.

Workload로는 Cornell Theory Center (CTC) SP2 trace로부터 synthetic workload를 생성하였다. 전체 workload는 200개의 병렬 작업들로 구성되어 있고, 각 병렬작업은 computation, I/O, communication을 반복적으로 수행한다. 병렬작업의 통신 패턴으로는 Nearest-Neighbor (NN)와 All-to-All (AA) 를 사용하였다. 또한 이상적인 종단간 지연시간을 바탕으로 computation, communication, I/O에 필요한 시간을 계산하고, 이를 바탕으로 병렬작업에 필요한 loop의 회수를 계산하였다. 표2는 본 연구에서 사용한 synthetic workload의 특성을 보여준다. 실험은 NOW 시스템에서 각 작업의 computation과 I/O 비율 (부하 불균형의 정도)인 Variance (v) 값을 변화시켜가면서 수행하였다.

작업 특성 (%)				workloads	
작업 타입	comp	I/O	comm		
J1	70	5	25	SWL1	J1
J2	40	20	40	SWL2	J2
J3	25	5	70	SWL3	J3

표 2. Synthetic Workload의 특성

4.2 실험 결과

성능 비교 수단 (performance metric)으로 본 연구에서는 평균 작업 반응 시간을 사용하였다. 또한 시스템이 실제 computation, IDLE, spinning, 문맥교환에 소비한 시간과 동시 스케쥴링을 수행하기 위해서 필요한 오버헤드 (인터럽트 처리 시간, 부하 정보 계산 및 프로세스 재순서 알고리즘 수행 비용 등)의 비율을 보여주는 시스템 분석 데이터를 사용하였다.

그림 4는 기존의 SB, PB 기법에 PROC기법을 추가적으로 적용시켰을때의 성능의 보여준다. 여기에서는 MPL은 5, v는 0.5로 고정시켰다. 그림에서 보는 바와 같이 모든 통신 패턴과 작업 특성에서 PROC 기법을 적용한 것이 더 나은 성능을 보이고 있음을 알 수 있다 (최대 21.4%까지 평균 작업 반응 시간 감소). 특히 SB와 같이 메시지 기다림 행동이 블록 기반의 방법에서는 SB+RO이 SB에 반해 더 적은 IDLE 시간(최대 55.6%까지 감소)을 소비함을 알 수 있다. 또한 PB와 같이 기다림 행동이 spinning인 경우에는 PB+RO이 PB에 비해 더 적은 SPIN 시간 (최대 31.8%까지 감소)을 소비함을 알 수 있다. 그리고 전체적으로 PROC에서는 부하 수준의 계산 및 재순서 알고리즘의 적용으로 인해 오버헤드와 문맥 교환 비용이 더 증가함에도 불구하고 SPIN 및 IDLE 시간을 많이 줄이기 때문에 더 나은 평균 작업 반응 시간을 보인다.

그림 5는 Variance값을 변화시켰을 때 기존의 기법과 PROC의 성능을 비교한 것이다. 그림에서 보는 바와 같이 v=1.5일때(즉, 부하 불균형이 클 때)는 통신하는 프로세스들간의 동기화 확률이 그만큼 더 감소함으로 v=0.5일때와 비교해서 작업 반응 시간이 더 늘어남을 알 수 있다. 또한 PB+RO와 SB+RO 기법이 PB와 SB에 비해 각각 ??%와 ??%까지 평균 작업 반응 시간을 줄임을 알 수 있다.

본 논문에서는 NOW 시스템에서의 부하 불균형을 고려하여 프로세스의 재순서를 이용한 동시 스케쥴링 기법인 PROC을 제안하고, 그 우수성을 시뮬레이션을 통해 검증하였다. 앞으로는 더욱 현실적인 workload (NAS Parallel 벤치마크 등)들을 모델링하여 실험을 수행하고, 제안한 PROC기법을 리눅스 클러스터에서 구현할 계획이다.

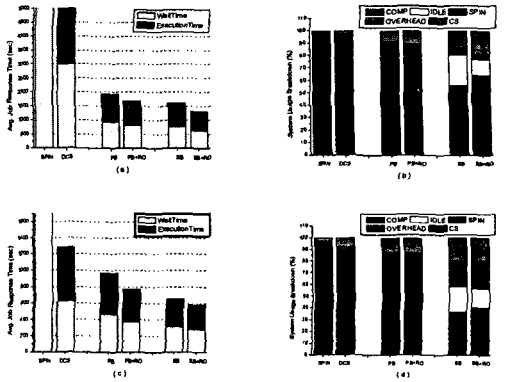

그림 4. PROC 기법의 성능 (MPL=5, v=0.5) : (a)(b) NN-SWL1의 평균 작업 반응 시간과 시스템 사용 비율, (c)(d) AA-SWL3의 평균 작업 반응 시간과 시스템 사용 비율

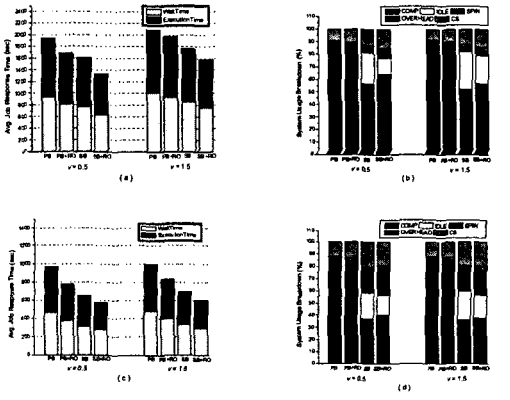

그림 2. 부하 불균형의 영향 (MPL=5) : (a)(b) NN-SWL1의 평균 작업 반응 시간과 시스템 사용 비율, (c)(d) AA-SWL3의 평균 작업 반응 시간과 시스템 사용 비율

참고 문헌

[1] C. Anglano, " A Comparative Evaluation of Implicit Coscheduling Strategies for Networks of Workstations," HPDC 2000, pp. 221-228

[2] U. Rencuzogullari and S. Dwarkadas, " Dynamic adaptation to Available Resources for Parallel Computing in an Autonomous Network of Workstations," PPPP 2001, pp. 72-81

[3] H. D. Schwetman, " CSIM19: a powerful tool for building system models," 2001 Winter Simulation Conference, pp. 250-255