IBM LoadLeveler에서 Preemption 기능을 적용한 큐 구성

이영주, 성진우, 김성준, 박찬열
한국과학기술정보연구원
e-mail:yjlee@kisti.re.kr

The Queue Implement Using Preemption in IBM LoadLeveler

Young-Joo Lee, Jin-Woo Sung, Sung-Jun Kim, Chan-Yeol Park
Korea Institute of Science Technology Information

요 약

하나의 시스템을 다수의 사람들이 작업을 할 경우 한정된 자원을 각각의 작업에 효율적으로 배분하기 위하여 작업관리 시스템을 이용한다. IBM 시스템은 작업관리 시스템으로서 주로 LoadLeveler를 사용하고 있다. 작업관리 시스템은 작업을 처리할 수 있는 여러가지의 큐를 가지고 있으며, 큐는 시스템의 특성과 구성 그리고 사용자 작업의 패턴에 따라서 설계되어진다.

본 논문에서는 LoadLeveler에서 작업을 실행할 때 긴급한 작업을 우선적으로 처리할 수 있는 큐를 만들기 위하여 preemption 기능을 이용하여 구성하고, 해당 큐로 작업을 선점하여 실행하면서 각각 작업들의 메모리 사용 변화와 그에 따른 작업 처리 성능을 분석하고 방법을 연구하였다.

1. 서론

작업관리 시스템은 한정된 전체 시스템의 자원을 다수의 사용자가 요구한 자원에 따라서 효율적으로 배분하고 관리할 수 있는 유틸리티이다. 이러한 작업관리 시스템은 전체 시스템 자원을 하나의 사용자가 독점 사용하는 것을 막고 동시에 다수의 사용자가 각각의 프로그램에서 처리할 자원을 요구할 때 각각의 요구자원을 배분한다. 작업관리 시스템은 여러 종류가 있으며 시스템의 종류와 작업의 특성에 따라서 적당한 작업관리 시스템을 선택하여 사용하고 있다. KISTI에 설치된 IBM p690 시스템에서는 LoadLeveler, NEC SX-5/6 시스템은 NQS, 하멜 클러스터 시스템은 PBS 등을 사용하고 있다. 작업관리 시스템은 전체 시스템 자원을 각각의 작업에 분배하고 처리하기 때문에 작업관리 시스템의 종류와 작업 실행 큐 구성에 따라 작업처리의 성능에 많은 영향을 준다. 작업관리 시스템의 구성을 설계하고 관리하는데 여러 가지 많은 환경변수를 가지고 있지만 그 중에서도 CPU와 메모리가 가장 큰 환경 변수이다.

본 논문에서는 작업관리 시스템을 통하여 작업을 실행하는 중에 긴급하게 처리할 작업을 위한 realtime 큐를 preemption 기능을 이용하여 구성하고, 해당 큐를 이용하여 작업을 실행하면서 작업들에 대한 메모리 사용 변화와 그에 따른 작업 처리 성능을 분석하였다.

2. 관련 연구

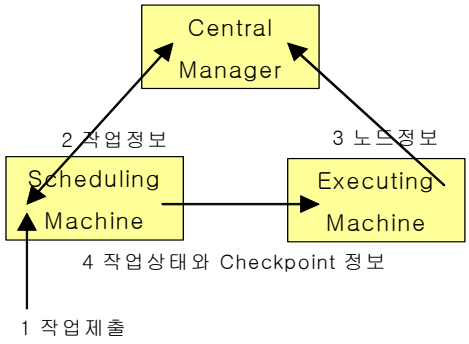
2.1 LoadLeveler

LoadLeveler는 분산컴퓨터를 위한 작업로드 관리 시스템으로서 사용자가 여러 노드 또는 시스템을 하나의 컴퓨터처럼 사용할 수 있게 한다. 또한 여러 시스템의 작업 부하를 균형 있게 관리하며, 순차 프로그램과 병렬 프로그램을 모두 지원한다. 이 작업관리 시스템은 원래는 위스콘신 대학교의 Condor Job Scheduler를 기초로 한 것으로 IBM에 의해 사용자 기반 우선 순위, NFS/AFS 지원, GUI 등의 기능이 추가되었다.

2.2 LoadLeveler 구조

LoadLeveler는 사용자의 프로그램을 실행하기 위해 명시한 요구자원을 시스템 자원으로부터 할당받아 처리한다. 사용자가 LoadLeveler를 통하여 작업을 제출하면 작업은 일단 LoadLeveler에 대기하다가

해당 작업의 처리 조건이 가능한 클래스를 찾으면 해당 큐에 작업을 할당하여 실행한다.



(그림 1) LoadLeveler job cycle

2.3 Preemption 형태

작업관리의 스케줄링 방법으로서 작업을 선점하는 기능으로서 LoadLeveler에서는 두가지 방식을 제공한다. 하나는 시스템에서 자동적으로 실행하게 하는 것이고 다른 하나는 사용자의 의해서 수동적으로 실행하는 것이다. 선점 방법은 <표 1>과 같이 3가지정도로 나눌 수 있다.

<표 1> 자원관리 방법의 형태

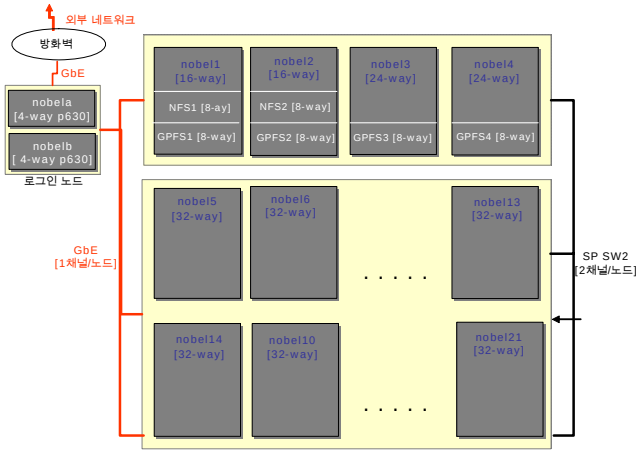
구분	내 용	비 고
Job Requeue	실행 중인 작업을 종료하고 큐가 쉬고 있을 때 작업을 다시 시작한다	작업 중단으로 CPU 낭비가 발생함
Job Suspend	실행 중인 작업을 멈추게 한다. 그러나 컴퓨터 노드의 할당이나 메모리는 그대로 유지한다	메모리 낭비가 발생함
Job Checkpoint	실행 중인 작업의 현재까지의 상태를 저장한다. 체크포인트 된 작업이 재실행 될 때에는 가장 최근의 체크포인트로부터 시작한다	작업 중단 없음, 단 시스템에서 지원 가능해야 함

3. 시스템 환경 구성

3.1 IBM 시스템 구성

KISTI의 IBM p690 시스템은 (그림 2)와 같이 모두 21노드로 이루어진 IBM p630, p690 시스템이다. O/S는 AIX 5.2이고 작업관리 시스템은 LoadLeveler 3.3.2 이다. p630 시스템은 주로 로그인 등 서버로 사용하고, p690 시스템은 계산노드로 사용하고 있다. 하나의 노드는 32CPU로 구성되어 있다. 홈디렉토리는 NFS로 연결되고 사용자의 작업을 위한 스크래치 디렉토리는 GPFS를 통하여 연결되어 있다.

본 논문에서 실험 대상으로 하는 시스템은 이 중에 두 노드를 선택 사용하였다. 메모리는 노드당 256GB로서 각각의 CPU에서 공유한다. paging space는 각각 57GB이다.



(그림 2) IBM 시스템 구성도

3.2 LoadLeveler 클래스 구성

LoadLeveler에서 큐를 클래스라고 부르고 클래스는 <표 2>과 같이 크게 4가지 유형 시스템에 따른 p630, p690과 작업의 형태에 따른 순차, 병렬로 나누어 구성하였다. 작업 실행 중 긴급하게 처리하여야 할 realtime 큐를 구성하기 위하여 p_realtime과 p_economy 큐를 만들었다. p_realtime 큐는 p_economy realtime보다 우선 순위가 높고 p_economy 큐를 선점할 수가 있다.

<표 2> LoadLeveler의 클래스 구분

클래스 Name	Priority	최대 CPU	허용시간	비고
normal	1	160	1440	일반1CPU
p_normal	2	448	1440	일반병렬
p_normal_1.3	2	64	1440	p630
p_normal_1.7	2	384	1440	p690
bmt	2	64	730	지정노드
p_realtime	3	64	720	realtime
p_economy	0	64	720	지정노드

3.3 Preemption 환경 구성

Preemption의 기능 구성 옵션에는 여러 가지가 있지만 주요 옵션은 <표 3>처럼 크게 네 가지정도

로 나눌 수 있다. 첫 번째 옵션 DEFAULT_PREEMPT_METHOD는 su로 설정하였다. 이것은 Default 옵션으로서 preemption되면 Preempted state로 상태 변화되며 동일 노드 상에서 preemption 시킨 작업이 종료되고, 자원이 사용가능할 때 다시 재 수행된다. process tracking이 반드시 필요하다.

두 번째 옵션 PREEMPT_CLASS[]=ENOUGH[]은 클래스 간의 우선 순위는 incoming_class가 높다. ENOUGH의 경우, incoming_class와 outgoing_class 간의 자원경쟁이 발생할 경우에만 incoming_class가 필요로 하는 자원에 한해서 일부 job step들이 preemption 된다.

세 번째 옵션 PREEMPTION_SUPPORT = full은 Preemption을 사용하기 위한 설정이다.

네 번째 옵션 PROCESS_TRACKING은 Backfill suspend method에서는 반드시 true로 설정해야한다. 특정 작업 종료 시, 잔여 프로세스가 자원을 계속 사용할 수 있으며 이는 시스템 성능을 떨어뜨리고 새로 수행되는 작업의 hang-up이나 수행 실패를 유발한다. 이를 방지하기 위해 PROCESS_TRACKING을 true로 설정해서 작업이 종료될 때, 그 작업과 관련된 모든 프로세스들을 감시하여 종료시키게 한다.

<표 3> preemption 설정 주요 옵션

DEFAULT_PREEMPT_METHOD=rm|sh|su|vc|uh
PREEMPT_CLASS[incoming_class]=ENOUGH{outgoing_class}
PREEMPTION_SUPPORT = full
PROCESS_TRACKING = true | false

4. Preemption 기능을 적용한 작업 실행 테스트

4.1 프로그램 실행 환경

작업 수행 테스트는 두 개의 노드를 이용하여 작업을 실행하였다. 각각 노드의 시스템 메모리는 250GB이다. 여기에서 실행할 프로그램을 <표 4>와 같이 구성하였다.

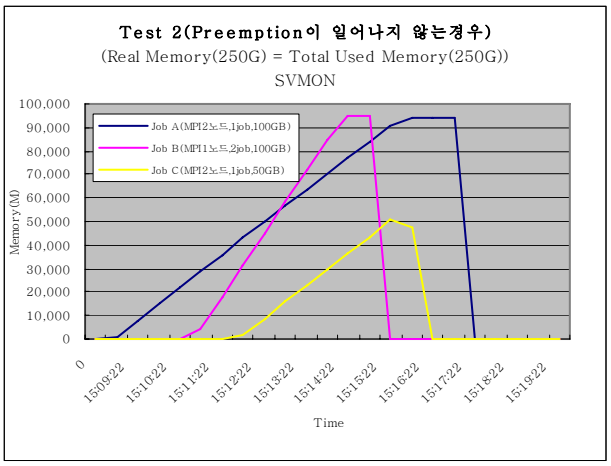
각각 작업의 처리 결과에 대한 성능 측정을 위한 툴은 svmon과 nmon을 사용하였다. svmon은 각각 사용자에 대한 모니터링이 가능하고, nmon은 전체적인 CPU, 메모리 등의 성능 측정을 그래프로 나타내서 보여준다.

<표 4> 메모리가 초과되지 않는 작업 환경

구분	클래스	메모리(GB)	작업형태
Job A	bmt	100	MPI
Job B	p_economy	100	MPI
Job C	p_realtime	50	MPI

4.2 시스템 메모리가 충분한 경우의 preemption

(그림 3)은 bmt, p_economy, p_realtime 순으로 하나씩 순차적으로 작업을 보냈을 경우 작업 전체의 메모리가 시스템 메모리를 초과하지 않기 때문에 page in/out이 발생하지 않고 각각의 작업이 정상적으로 종료되었다.



(그림 3) 메모리가 충분할 경우 작업 실행 결과

<표 5>는 3개의 작업이 각각 시작되어서 preemption 되어 정상적으로 작업이 종료된 것을 보여주고 있다. 작업의 실행 시간은 p_economy와 p_realtime은 각각 약 5분 bmt 클래스는 약 13분도 소요되었다.

<표 5> preemption 경우 작업처리 시간

	p_realtime (50GB)	p_economy (100GB)	bmt (100GB)
Real time	00:04:56	00:05:05	00:13:01
User time	00:03:38	00:03:35	00:07:16
Sys time	00:01:05	00:01:22	00:04:13

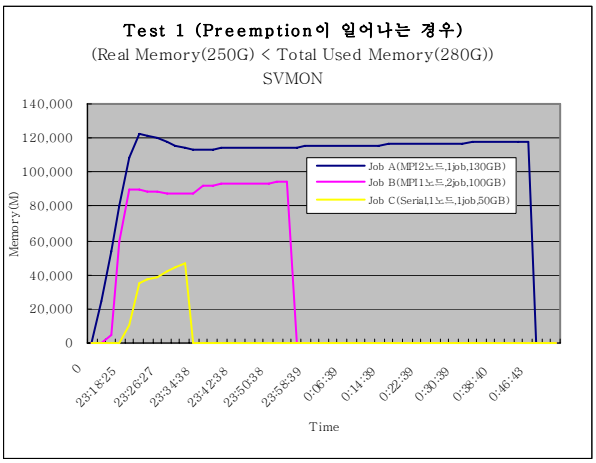
4.1 시스템 메모리가 부족한 경우의 preemption

preemption 할 작업이 시스템의 남은 메모리보다 큰 메모리의 작업을 보낼 경우, 이 때 작업들 간의 preemption이 이루어지는 과정과 메모리 상태 및 실행 시간을 측정하였다.

<표 6> 메모리가 초과되는 작업 환경

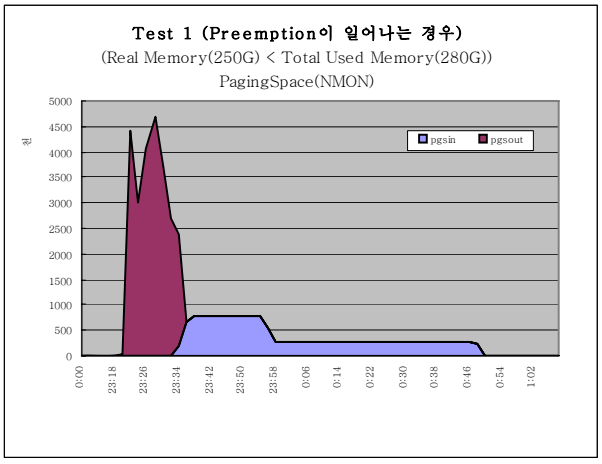
구분	클래스	메모리(GB)	작업형태
Job A	bmt	130	MPI
Job B	p_economy	100	MPI
Job C	p_realtime	50	MPI

(그림 4)는 3개의 작업이 각각 시작되어서 preemption 되어 작업이 종료된 것을 보여주고 있다. (그림 3)에서 각각의 작업들이 시작하여 정상적으로 종료되는 것과는 다르게 preemption 하는 작업은 정상적으로 종료되었으니 preemption 당한 두 작업들의 실행 시간이 지연되는 것을 보여주고 있다.



(그림 4) 메모리가 불충분할 경우 작업 실행 결과

(그림 5)는 기존의 수행 중인 작업에 시스템 메모리보다 큰 작업을 preemption 할 경우 수행 중인 작업들이 메모리 확보를 위하여 page in/out을 많이 하고 있는 것을 작업의 진행 순서대로 알 수 있다.



(그림 5) preemption 경우 메모리 Paging 변화

<표 7>을 보면 수행 시간이 약 5분 정도 걸리는 p_realtime과 p_economy 클래스 작업은 각각 17분과 43분 그리고 약 13분 정도 걸리는 bmt 클래스 작업은 1시간 40분이 소요되었다. 이것은 preemption 된 작업이 필요한 메모리 확보를 위하여 빈번한 page in/out 때문에 실제 작업 수행 시간이 지연되기 때문이다.

<표 7> preemption 경우 작업처리 시간

	p_realtime (50GB)	p_economy (100GB)	bmt (130GB)
Real time	00:17:26	00:42:51	01:39:41
User time	00:03:38	00:06:35	02:44:16
Sys time	00:01:05	00:03:22	00:08:13

5. 결론

배치 큐에서의 작업 실행 시 realtime 큐를 두어 긴급한 작업을 실행한 결과 realtime 큐가 잘 동작하는 것을 확인하였다. 하지만 realtime 큐 작업의 메모리가 시스템 메모리보다 클 경우 다른 큐에서의 작업 실행시간이 많이 지연되는 것을 알 수 있었다. 이것은 realtime 큐의 작업이 다른 작업의 메모리를 잠식하여, 이로 인한 메모리 부족으로 빈번한 page in/out 발생하기 때문이다. 일반적으로 시스템의 성능은 메모리 구조나 속도가 가장 큰 영향을 주기 때문이다.

따라서 realtime 큐로 작업을 실행 시 메모리를 고려하여 사용한다면 realtime 큐의 성능을 충분히 활용할 수 있다.

향후에는 이러한 문제점을 보완하기 위하여 reservation 기능을 추가로 적용하여 preemption을 이용한 realtime 큐를 구성하여 테스트하고자 한다.

참고문헌

[1] IBM, LoadLeveler Using and Administering
[2] IBM, AIX Workload Manager.
[3] IBM, LoadLeveler Guide
[4] IBM, AIX Performance Manager
[5] 이영주 외 "NEC 시스템의 효율적인 활용을 위한 작업관리 큐 설계 및 구현" 한국정보처리학회 2005 춘계학술대회
[6] KISTI, IBM System User Guide, 2006