제31회 한국정보처리학회 춘계학술발표대회 논문집 제16권 제1호 (2009. 4)

중요도에 따른 분산 로그분석 스케줄링

백봉현*, 안병철*
*영남대학교 컴퓨터공학과
e-mail : wefbbh@yu.ac.kr, b.ahn@yu.ac.kr

A Study on Scheduling of Distributed Log Analysis by the importance of the measure

BongHyun Back*, Byoungchul Ahn*
*Dept. of Computer Science, YeoungNam University

요 약

이기종(異機種) 시스템환경에서 발생하는 수많은 로그 데이터는 중요도에 따라 실시간 로그 분석이 필요하고 대용량의 로그 데이터의 경우 특정 시간내에 로그 분석 처리를 종료해야만 한다. 보안에 관련된 로그 정보의 경우 실시간 분석과 빠른 통계 처리를 요구할 것이며, 대용량의 비실시간성 로그 분석의 경우 로그 분석 및 통계처리를 주어진 특정 시간 내에 하여야 한다. 본 논문에서는 로그 데이터의 중요도에 따른 실시간 로그 분석 처리와 비실시간 대용량 통계 로그의 로그 분석 처리 마감 시간을 충족하는 로그 분석 스케줄링 정책을 제안한다.

1. 서론

로그는 컴퓨터의 이용상태 등의 기록을 남기는 데이터를 지칭하며, 최근 수 많은 기업들은 자사(自社)의 시스템으로부터 출력되는 여러 종류의 로그분석을 통하여 모니터링 및 레포팅하고 있다. 로그 데이터 분석의 문제점으로는 다기종(多機種)의 시스템으로부터 출력되는 상이한 로그형식과 그러한 시스템에 의존적인 텍스트 로그 파일은 검색 조건 및 집계방식의 차이로 검색 및 집계가 어렵다. 또한 자료의 중요도에 따른 로그의 분석 처리가 이루어지고 있지 않다는 것이다.

본 논문에서는 실시간을 요하는 중요도가 높은 로그와 실시간을 요하지 않지만 마감시간을 지켜야 하는 대용량 통계 로그들을 중요도에 따른 스케줄링 정책을 사용하여 실시간 로그 분석 및 로그 분석 처리 마감 시간을 충족하는 스케줄링 정책을 제안한다.

2. 관련연구

2.1 로그 분석

인터넷의 급속한 발전과 보급 및 손쉽게 구할 수 있는 해킹툴들의 발달로 원본 데이터 및 로그 데이터가 변질되어 많은 피해가 증가되고 있다. 또한 원본 로그 및 로그 분석 결과는 문제 발생 후 사후(事後) 처리시 법적인 증거로 이용이 가능하다. 침해 발생 후 이를 감지하고 추적하는 대응에 있어서 분석될 다양한 로그 데이터의 통합관리가 어렵고 실시간으로 분석된 로그는 사용자 및 유해공격으로부터 시스템 침입시도와 시스템의 자원관리가 가능할 수 있도록 하기 위해 로그분석이 필요 시 된다.

2.2 실시간 로그

실시간 로그 분석은 일반적으로 발생되는 로그 데이터를 발생과 동시에 분석 처리하는 것을 의미로 한다. 네트워크 보안과 관련하여 발생하는 로그 데이터 및 각종 시스템에서 발생하는 긴급 처리를 요하는 경우에 실시간 분석 처리를 필요로 한다.

2.3 비 실시간 로그

긴급 사항 이외의 로그 데이터를 분석하는 것으로 빠른 로그 데이터 분석 처리가 필요하지 않을 경우에 어플리케이션 상에서 보존 후 데이터를 로그 분석 서버로 이동 후 처리하는 하는 것으로 말한다.

어플리케이션에 보존된 로그 데이터는 ftp 와 ssh 등의 네트워크를 통해 로그 파일을 분석 서버로 전송 받는다. 전송된 로그 데이터는 특정 시간내에 분석처리를 완료하여야 한다.

2.4 리눅스 실시간 스케줄링 알고리즘

리눅스에서 실시간 알고리즘은 First-In First-Out 과 Round Robin 이 있다. First-In First-Out 은 먼저 들어온 것이 먼저 나가는 방식의 실시간 프로세스이다. 스케줄러는 프로세스에 CPU 를 할당할 때 실행 큐 리스트에서 해당 프로세스 디스크립터를 현재 있는 위치에 그대로 놔둔다. 우선순위가 더 높은 실행 가능한 실시간 프로세스가 없다면, 우선순위가 동일한 실행 가능한 다른 프로세스가 있더라도 자신이 원하는 동안 계속해서 CPU 를 사용한다. Round Robin 방식은 스케줄러가 프로세스에게 CPU 를 할당할 때 해당 프로

세스 디스크립터를 실행 큐 맨 끝에 넣는다. 이 정책은 똑같은 우선순위가 동일한 Round Robin 실시간 프로세스 사이에서 공정한 CPU 시간 할당을 보장한다.

2.5 리눅스 비실시간 스케줄링 알고리즘

리눅스의 비실시간 스케줄링 알고리즘은 시분할 스케줄링 알고리즘을 사용한다[4]. CPU 시간은 '시기(epoch)' 단위로 나누어 동작한다. 한 시기 동안 모든 프로세스는 정해진 타임 퀀텀을 소유하며, 이 기간을 그 시기가 시작할 때 계산한다. 타임 퀀텀 값은 그 시기 동안 프로세스에 할당하는 최대 CPU 시간이다. 프로세스가 타임 퀀텀을 모두 소비하면 해당 프로세스를 선점하여 다른 실행할 수 있는 프로세스로 교체한다. 물론 프로세스가 퀀텀을 모두 소비하지 않는 한 스케줄러는 한 시기 안에 그 프로세스를 여러 번 선택하여 실행할 수도 있다. 한 시기는 실행 가능한 모든 프로세스가 자신의 퀀텀을 완전히 소비할 때 끝난다. 이 경우 스케줄러 알고리즘은 모든 프로세스 타임 퀀텀을 다시 계산하고, 새로운 시기를 시작한다.

2.6 Rate Monotonic 알고리즘

연결 설정 과정에서 작업들의 요청 비율에 따라 정적인 우선순위 작업을 설정한다. 계속해서 각각의 작업들은 우선순위 요청에 따른 재조정 없이 처음에 계산된 우선순위에 의해서 처리된다. 우선순위는 다른 작업들에 대한 작업의 중요도와 같다.

RM 알고리즘에서 가장 짧은 처리 시간을 가지는 작업이 가장 높은 우선순위를 갖고, 가장 긴 처리 시간을 가지는 작업이 가장 낮은 우선순위를 갖는다. RM 은 최적화 된 정적인 우선순위 기반 알고리즘이다.

2.7 least Laxity First 알고리즘

LLF(Least Laxity First)는 현재 시간과 마감시간을 뺀 시간에 처리시간을 뺀 값이 가장 작은 프로세스를 CPU 가 선택한다. 즉 앞으로 실행 되어야 할 시간이 가장 작은 프로세스를 CPU 는 선택한다. LLF 는 독점적인 자원에게 최적화되고 동적인 알고리즘이다. 만약 실시간 프로세스의 준비 시간이 같다면 다수의 자원에게 최적화 되어 있다. Laxity 는 마감시간, 처리시간, 현재시간으로 정해진다. 그러나 처리시간은 미리 정해지지 않기 때문에 최악의 경우에는 laxity 가 부정확 할 수 있다. 그리고 기다리는 프로세스의 laxity 는 계속 변화하며 실행하는 동안 실행중인 프로세스를 선점하기 위해 다른 프로세스는 낮은 laxity 가 필요하다. 결과적으로 프로세스는 새로운 프로세스가 생성되지 않아도 여러 번 선점되며 이러한 점은 빈번한 문맥교환을 야기 시킨다.

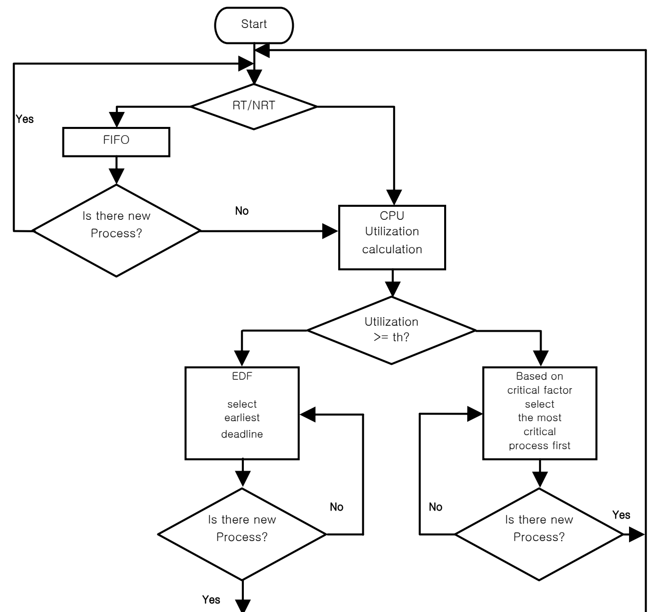
2.8 Earliest Deadline First 알고리즘

EDF 알고리즘은 마감시간이 가장 촉박한 태스크를 먼저 실행한다. 어떤 새로운 작업이 도착했을 때, 새로운 스케줄링 순서를 즉시 계산한다. 즉, 현재 진행중인 태스크의 마감시간과 새로운 작업의 마감시간을 비교하여 새로운 작업의 마감시간이 더 촉박한 경우 실행되고 있는 작업은 새로운 작업에 의해 선점된다[5, 6]. EDF 는 최적화 된 동적 알고리즘이다. 프로세스의 CPU 이용률이 100%를 넘지 않으면, 프로세스 집합은 EDF 알고리즘을 이용하여 스케줄링 가능하다. EDF 는 주기적인 작업들뿐만 아니라, 비주기적인 작업들, 제한 시간과 서비스 실행 시간을 갖는 작업들도 처리한다. 그러나 프로세스들의 CPU 이용률의 합이 100%를 넘어서는 과부하 상태에서 EDF 는 어떠한 작업의 처리도 보장할 수 없다[7].

3. 제안 알고리즘

제안된 알고리즘은 로그 데이터의 실시간 처리 및 비실시간 처리 로그로 구분하는 중요도에 따른 우선순위 기반의 알고리즘을 선택하는 방식을 제안한다.

알고리즘을 설명하기에 앞서 어느 시점에 발생할지 모르는 실시간의 로그 데이터를 최상위 우선순위로 하고 비실시간의 대용량 로그 데이터를 하위순위로 채택한다. 실시간과 비실시간의 로그가 발생 시 우선순위를 실시간 로그가 처리되도록 하며, 실시간 로그는 발생되어진 로그의 분석을 완료하기 전까지는 CPU 를 선점하도록 한다. 실시간 로그의 발생이 없을 때는 비실시간 로그 처리가 CPU 를 선점하도록 하며, 비실시간 로그의 처리도중 실시간 로그가 발생할 경우 다시 실시간 로그 처리가 CPU 를 선점하도록 한다.



(그림 1) 제안 알고리즘

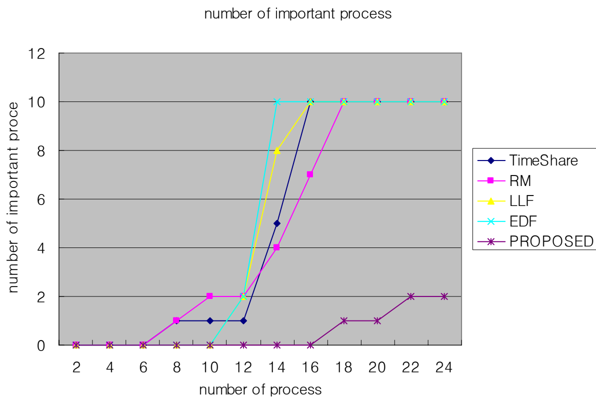
4. 성능측정 및 분석(비교)

본 논문은 제안된 알고리즘을 CHEDDAR 시뮬레이터를 사용하여 각각의 알고리즘들을 평가하였다[9].

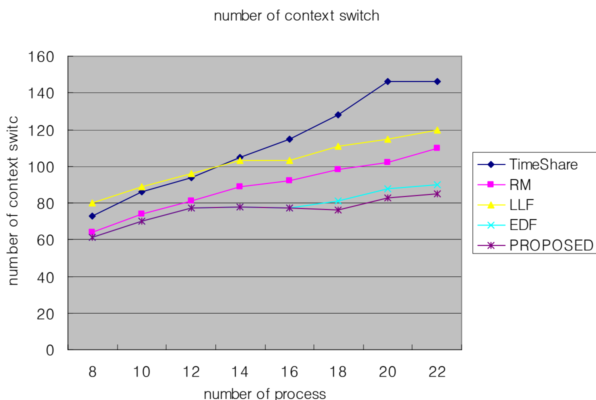
성능 측정은 세가지로 나누어 측정하였다. 첫 번째는 각각의 알고리즘에 대해 프로세스 수를 증가시키면서 마감시간을 위반하는 프로세스의 수와 문맥교환

의 수를 측정하였고, 두 번째는 대용량 로그의 프로세스 수를 증가시키며 마감시간 위반 수를 측정한다. 세 번째는 중요도를 가진 프로세스를 정해놓고 프로세스의 수가 증가함에 따라 각각의 중요도를 가진 프로세스들이 마감시간을 위반한 수를 측정한다.

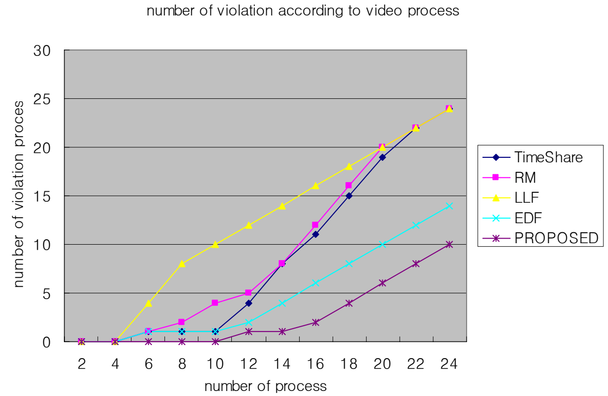
다음은 각각의 알고리즘들에 대해 프로세스 수를 증가시키면서 마감시간을 위반하는 프로세스의 수를 측정 한 결과이다.



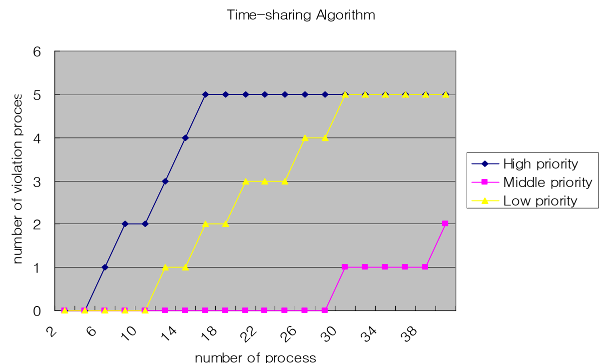
프로세스 수가 증가함에 따라 중요한 프로세스의 마감시간 수를 측정하였다. 시분할 스케줄링 알고리즘과 RM 알고리즘은 프로세스의 수가 8 일 때 마감시간 위반을 시작하였다. 이 두 가지의 알고리즘은 CPU 이용률이 1 보다 적음에도 불구하고 스케줄러의 정책에 의해서 가장 먼저 알고리즘을 위반하였다. EDF 나 LLF 알고리즘은 프로세스의 수가 12 이상으로 넘어가면서부터 마감시간을 위반하는 프로세스의 수가 급격하게 증가하기 시작하였다. 제안된 알고리즘은 프로세스의 수가 18 부터 마감시간을 위반하였으며 안정적인 것으로 판단된다. 프로세스의 수가 18 부터 위반한 이유는 중요한 프로세스들 간에 CPU 이용률이 1 을 넘었기 때문이다. 다음의 각각의 알고리즘에 따른 문맥교환 수를 나타내었다.

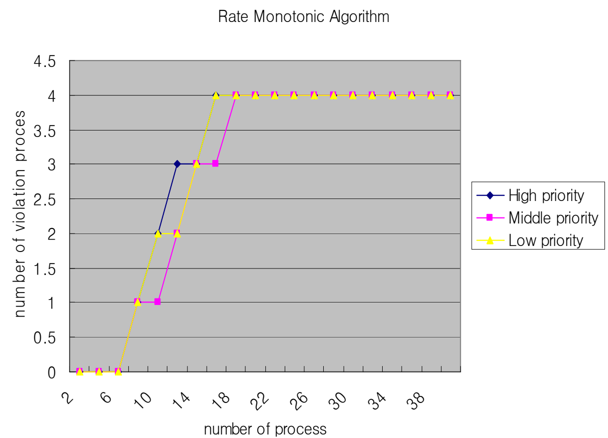


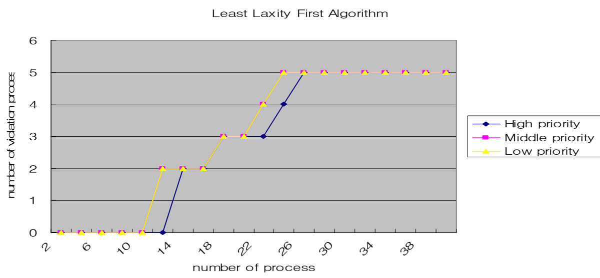
시분할 스케줄링 알고리즘과 제안된 알고리즘은 10%의 차이를 보이나, 프로세스의 수가 증가함에 따라서 60% 또는 그 이상의 차이를 보인다. 제안된 알고리즘과 EDF 는 거의 비슷한 모습을 보이지만 제안된 알고리즘의 문맥교환 수가 약간 작을 모습을 보여주고 있다. 그 이유는 CPU 이용률이 Threshold 값을 넘기 전에 제안된 알고리즘은 EDF 알고리즘 방식을 사용하다가 CPU 이용률이 Threshold 값을 넘은 후에는 중요한 프로세스를 먼저 처리하기 때문에 문맥 교환 횟수가 차이를 보인다. 다음은 대용량 로그 처리 프로세스 수를 증가시켜 마감시간 위반 수를 측정한 결과이다.

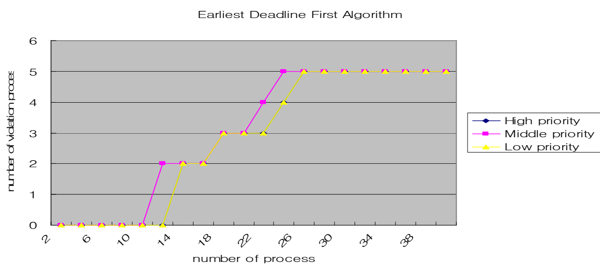


제안된 알고리즘은 다른 스케줄링 알고리즘 보다 마감시간을 위반하는 프로세스의 수가 2 배정도 차이가 남을 볼 수 있으며, 위반 시기가 다른 프로세스에 비해 늦게 나타남을 보인다. 다음은 각각의 다른 중요도를 가진 프로세스 수를 증가 시킴에 따라 각각의 중요도를 가진 프로세스들이 마감시간을 위반한 수를 측정한 결과 이다.

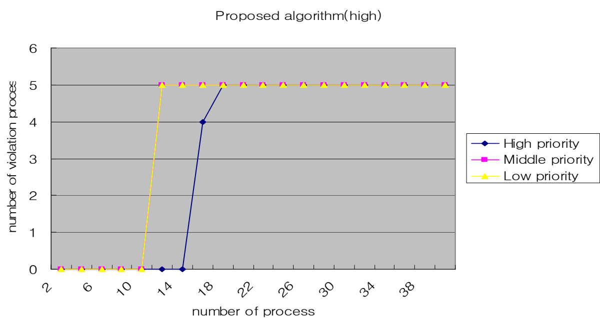


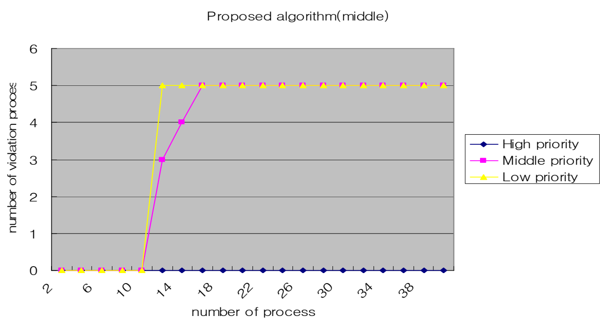


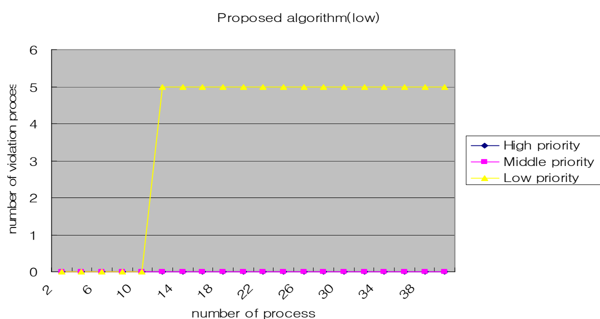




기존의 스케줄링 알고리즘은 중요도를 고려하지 않기 때문에 중요도에 관계없이 동일한 결과를 보여주며 중요 로그 프로세스의 처리를 고려하지 않았다.







첫 번째 그림은 높은 중요도를 가진 프로세스를 증가시켰을 때의 실험 결과이다. 모든 프로세스가 일정 프로세스 입력 후에 마감시간을 위반하였다. 그러나 높은 중요도의 프로세스는 마감시간 위반이 늦음을 알 수 있다. 이것은 제안된 알고리즘이 각각의 프로세스의 중요도를 고려한 결과로 볼 수 있다. 두 번째 그림은 중간 중요도, 세 번째 그림은 낮은 중요도를 가진 프로세스를 증가시켰을 때의 실험 결과이다. 위의 세가지 그림은 제안된 알고리즘이 높은 중요도를 가진 프로세스에게 최대한 많은 CPU 시간을 보장하는 것을 보여준다.

5. 결과 및 향후 과제

본 논문에서 제안된 알고리즘은 다른 알고리즘에 비해 중요한 프로세스의 마감시간 위반 수를 감소 시킬 수 있다. 이러한 결과는 CPU 이용률이 Threshold 값보다 작을 때는 EDF 알고리즘을 사용하여 CPU 의 이용을 최대화 하고, CPU 이용률이 Threshold 값보다 클 때는 중요한 프로세스를 선택하여 먼저 처리하므로 중요한 프로세스의 CPU 시간을 보장하기 때문이다. 문맥교환은 전체 계산 시간에 비해 상대적으로 매우 작은 시간이므로 스케줄러 설계 시 거의 무시되었다. 실험 결과에서 시분할 스케줄링 정책에 비해 최소 10%에서 최대 60%까지 문맥교환의 수를 감소 시킬 수 있었다.

참고문헌

[1] AWSD-WebLog. http://awsd.com/scripts/weblog/index.shtml.

[2] Diniel P. Bovet, Marco Cesati, Understanding the Linux Kernel, 2nd Edition, O' REILLY, September, 2003.

[3] C. L. Liu, J. W. Layland, "Scheduling Algorithm for Multiprogramming in a Hard Real-Time Environment," Journal of the ACM, 20-91, Jan, 1973.

[4] Ralf Steinmetz, "Analyzing the Multimedia Operating System," IEEE Multimedia, Spring, 1995.

[5] P. Goyal, X. Guo, H. M. Vin, "A hierarchical CPU scheduler for multimedia operating system," Proc. of the Symposium onOperating System Design and Implementation, October, 1996.

[6] J. Y. T Leung, M. L. Merrill, "A Note on Preemptive Scheduling of Periodic Real-Time Tasks," Information Processing Letters, PP.115, Nov, 1980.

[7] C. M. Krishna, Kang G. Shin, "Real Time System," The McGraw-Hill Companies, Inc, 1995.

[8] R. Steinmetz, "Analyzing the multimedia operating system," IEEE Multimedia, pp. 68, Spring, 1995.

[9] The cheddar project. "http://beru.univ-brest.fr/~singhoff/cheddar".