# UNAS: 메모리 활용과 성능 최적화를 위한 사용자 공간의 NUMA 인지 스케줄러

*임근식, $엄영익
*$성균관대학교, *삼성전자

{*leemgs, $yieom}@skku.edu, *geunsik.lim@samsung.com

# UNAS: User-Space NUMA-Aware Scheduler for Optimizing Memory Utilization and Performance

*Geunsik Lim, $Young Ik Eom
*$Sungkyunkwan University, *Samsung Electronics

요 약

멀티코어 CPU 는 애플리케이션의 고성능 처리를 위한 범용적인 시스템 구조로써 정착하였다. 이후 각 CPU 마다 메모리를 별도로 소유하는 NUMA 라는 메모리 아키텍처까지 진화하였다. 그러나, 사용자가 애플리케이션의 고성능 수행을 위해 NUMA 아키텍처를 충분히 이해하고 애플리케이션들을 실행해야 하기 때문에 진입 장벽이 매우 높다. 본 논문은 NUMA 아키텍처의 특성을 모니터링 한 후 태스크들을 위한 이상적인 메모리 영역을 자동으로 할당하는 사용자 공간의 스케줄러를 제안한다.

## Ⅰ. 서론

사용자들이 대용량의 계산 처리를 위한 고성능 컴퓨터가 필요한 것처럼 애플리케이션들은 항상 더 빨리 수행되기 위하여 고성능 시스템을 필요로 한다. 고성능의 요구는 멀티코어 아키텍처를 등장시켰다. 최근에는 Fig. 1 과 같이 각 CPU 마다 메모리를 소유하는 Non-Uniform Memory Architecture (NUMA)로 까지 진화하였다. 이로 인해 현대의 컴퓨터 설계는 쓰레드 스케줄링이라는 매우 도전적인 소프트웨어 과제를 만들고 있다. 사용자들은 메모리의 효과적인 활용과 성능 최적화를 위해서 깊이 있는 시스템적인 기술 지식이 필요하다. 사용자들이 NUMA 아키텍처 지식을 갖추어야 하는 진입장벽 때문에 NUMA 환경을 효과적으로 운영하는 것이 쉽지 않다. 따라서 본 논문은 불필요한 메모리의 지연을 제거하고 애플리케이션의 고성능 수행이 가능한 새로운 스케줄러를 제안한다. 우리의 제안 스케줄러는 사용자 공간에서 NUMA 아키텍처를 인지하여 메모리를 스케줄링 한다.

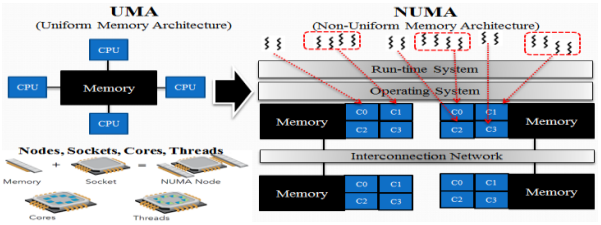


**Fig. 1 Evolution of memory architecture**

## Ⅱ. 관련 연구

**커널 공간의 NUMA 스케줄링 기법:** *SchedNUMA*[1]는 프로세스들이 동일한 NUMA 노드에 놓이도록 함으로써 NUMA 시스템에서의 메모리 지역성을 최적화한다. 그러나, 이 기법은 프로세스들을 그룹화 하기 위하여 NUMA 관련 추가 API 의 정의 및 구현이 필요하기 때문에 API 의 호환성을 유지하는 것이 어렵다.

*Automatic NUMA Balancing*[2]은 페이지 폴트 핸들러가 페이지 테이블 엔트리에 맵핑이 되어 있지 않은 캐시 페이지를 스왑핑할 때 태스크의 페이지들을 다른 메모리 노드로 이동시킨다. 본 기법은 커널 레벨에서 자동으로 메모리 스케줄링을 수행한다. 그러나, 최적의 NUMA 튜닝은 OS 에 의해서 수행되기 어렵기 때문에, 결국 시스템 관리자는 최적화 툴들을 이용하여 별도의 튜닝작업을 해야만 한다.

**사용자 공간의 NUMA 스케줄링 기법:** *Sergey Blagodurov* [3]는 NUMA 환경에서 성능 최대화를 위해서 사용자 공간에서 *CPU Affinity* 기법을[4] 이용하는 성능 최적화 기법을 제안하였다. 그러나 이 기법은 태스크들을 특정 NUMA 노드에 고정시키기 때문에 태스크들의 효과적인 메모리 활용을 손상시킨다. 더군다나 이 기법은 사용자 공간의 자동화된 메모리 스케줄링을 다루지 않는다.

## Ⅲ. 제안 시스템

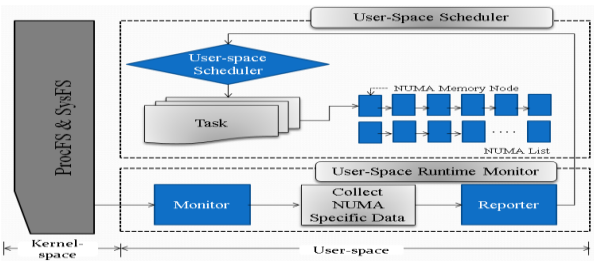


**Fig. 2 Architecture of our proposed system (UNAS)**

우리의 제안 기법은 불필요한 메모리 지연 가능성을 제거함으로써 고성능의 애플리케이션 수행이 가능하도록 이상적인 메모리 지역성을 만든다. 이를 위해서 제안 시스템은 NUMA 버스 토폴로지와 메모리 사용량을 모니터링 한 후 수집된 정보를 기반으로 하여 사용자 공간에서 태스크들에게 가장 이상적인 NUMA 노드에 찾아서 (재)할당시키는 작업을 자동으로 수행한다.

이 논문은 2013 년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보 컴퓨팅기술개발사업 의 지원을 받아 수행된 연구임(No.2010-0020730)

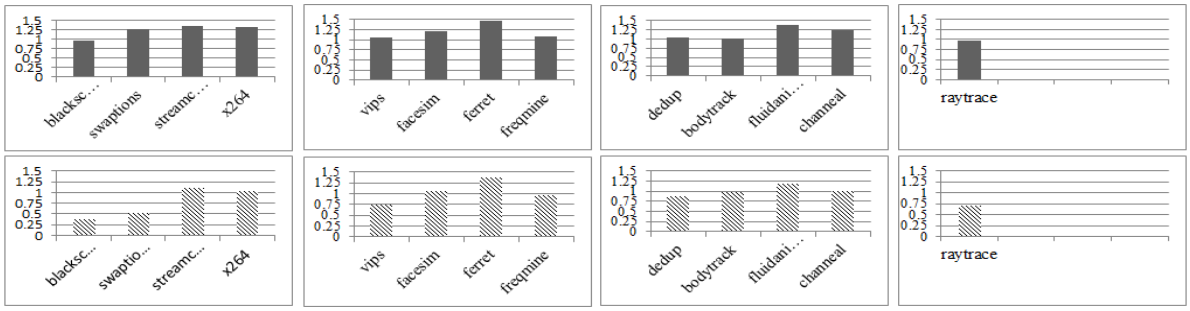


**Fig. 3 The accuracy of the performance degradation factor (upper figures refer to performance degradation due to contention; below figures refer to contention degradation factor.)**

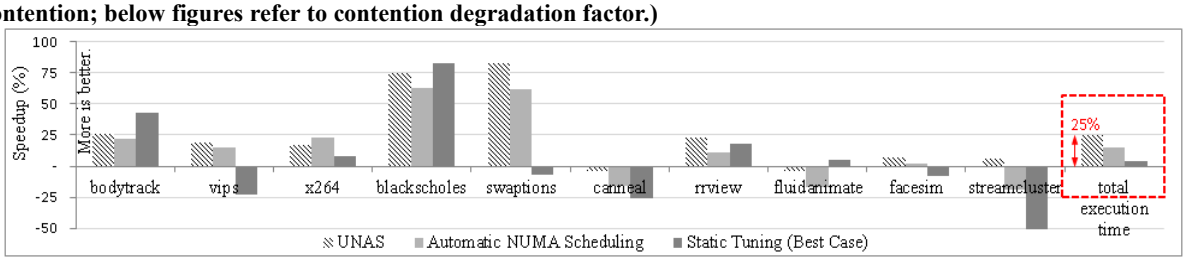


**Fig. 4 The speedup of our UNAS, the Automatic NUMA Scheduling, and the Static Tuning on 40-cores platform**

Fig. 2 는 NUMA 아키텍처의 특성을 모니터링 한 후에 사용자 공간에서 태스크들을 이상적인 메모리 영역으로 재배치시키는 시스템 아키텍처를 보여준다. 제안 시스템은 1) NUMA topology 기반의 메모리 사용량을 모니터링 하는 런타임 Monitor 와 2) 수집된 정보를 이용하여 사용자 공간의 스케줄러에게 NUMA 관련 데이터를 제출하는 Reporter, 3) 태스크들을 이상적인 메모리 노드에 (재)할당하는 사용자 공간의 Scheduler 로 구성된다. 제안된 시스템은 아래와 같이 3 개의 주요 알고리즘으로 동작한다. 알고리즘 1 은 NUMA 의 특성을 모니터링 하여 정보를 수집한다. 알고리즘 2 는 수집된 정보를 리포팅한다. 마지막으로 알고리즘 3 은 수집된 정보를 이용하여 쓰레드들을 이상적으로 메모리 노드에 (재)결합시킨다.

**Algorithm 1. Monitor: Runtime monitoring mechanism**

1. Create a new thread for receiving and dealing with the run-time monitoring data
2. **Repeat monitoring** until NUMA-aware user-space scheduler stop
3. Sleep for an NUMA specific data (from /proc/stat)
4. Collect the monitoring report
5. **End Repeat loop**

**Algorithm 2. Reporter: Collected NUMA specified data reporting mechanism**

**Input:** run-time monitoring data
1. **Repeat until** runtime monitoring mechanism stop
2. Receiving data and filtering them from online monitoring
3. Collect NUMA specific data
4. **If** loading of system is unbalanced or behavior of the processes changed or powerful core is idle
5. Computing the Run-time speedup factor
6. Sorting the process NUMA list by multi-core speedup factor
7. Computing the contention degradation factor
8. Sorting the process NUMA list by contention degradation factor
9. Sending signal to trigger schedule
10. **End if**
11. **End Repeat loop**

**Algorithm 3. User-space Scheduler: Automatic NUMA aware scheduling**

**Input:** NUMA list
1. Computing the number of powerful core candidate based on **load balanced memory policy**
2. Retrieving suitable processes to be scheduled on powerful cores from NUMA list
3. Setting static CPU pin from manual input of administrator
4. **If** retrieved processes != current processes on powerful cores
5. Migrate the processes
6. **End if**
7. **If** current resource contention degradation is too big
8. Scatter the processes with heavy contention
9. Calculating degradation factor in order to minimize resource contention degradation
10. Migrate the processes and the its sticky pages
11. **End if**

## Ⅳ. 평가

우리는 제안 기법의 성능 향상 효과를 검증하기 위해서 *40-core Intel Xeon Server (NUMA)*에서 실험을 하였다. 제안 아이디어의 평가는 워크로드에 집중하는 *PARSEC* 벤치마킹 프로그램으로 수행되었다. 우리는 워크로드의 절반은 CPU 집중적이고, 나머지 절반은 메모리 집중적인 프로세스들로 구축하여 실험하였다. Fig. 3 는 NUMA 아키텍처 환경에서의 메모리 경쟁으로 인한 성능저하와 성능저하 요인들의 상관 관계를 나타내고 있다. Fig. 4 는 기존 시스템과 제안 시스템의 애플리케이션 처리속도 개선 정도를 비교 실험한 결과이다. 실험 결과, 제안 시스템을 적용 시 전체 수행시간이 25% 개선되었다. Fig. 5 은 Apache 웹 서버와 MySQL 데이터베이스의 성능 개선 결과이다.

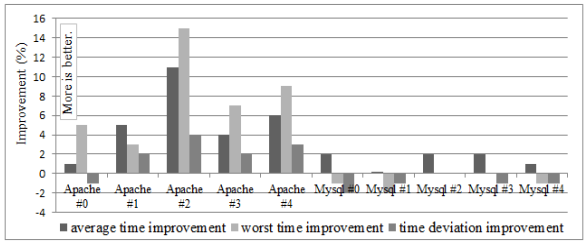


**Fig. 5 Performance experiment with Apache & MySQL**

## Ⅴ. 결론

제안 시스템은 사용자 공간에서 NUMA Topology 의 특성을 모니터링 한 후 수집한 정보를 이용하여 태스크들을 자동으로 이상적인 메모리 영역으로 (재)위치 시킨다. 제안 기법은 사용자 공간에서 태스크들의 메모리를 재배치시키기 때문에 운영체제의 수정이 필요하지 않다. 즉, 메모리 활용도와 성능 최적화를 모두 고려하는 새로운 사용자 공간의 메모리 스케줄러이다.

## 참 고 문 헌

[1] Peter Zijlstra, Red Hat, Sched NUMA Rewrite, https://lwn.net/Articles/508966/

[2] Mel Gorman, Google, Automatic NUMA Balancing, http://lwn.net/Articles/528881/

[3] Sergey Blagodurov et al., "A case for NUMA-aware contention management on multicore systems," ATC, 2011.

[4] Yinan Li et al., "NUMA-aware algorithms: the case of data shuffling," CIDR, 2013.