맵리듀스에서 리듀스 단계 성능 향상을 위한 적응적 리듀스 태스크 스케줄링 기법

이정하*, 최숙경*, 박지수*, 이은영**
*고려대학교 대학원 컴퓨터교육학과
**동덕여자대학교 컴퓨터학과
e-mail: jungha07@korea.ac.kr

Adaptive Reduce Task Scheduling Technique for Improving Reduce Phase in MapReduce

Jungha Lee*, SookKyoug Choi*, JiSu Park*, EunYoung Lee**
*Dept. of Computer Science Education, Korea University
**Dept. of Computer Science, Dongduk Women's University

요 약

맵리듀스는 데이터 집약적인 어플리케이션에서 대량의 데이터를 분산·병렬 처리하기 위한 프로그래밍 모델이다. 하둡은 맵리듀스의 오픈소스 구현으로 맵리듀스를 사용하기 위한 도구로 많이 알려져 있다. 실제 하둡을 이용하여 맵리듀스를 적용할 때 맵 태스크 단계는 병렬로 수행되어 순차처리에 비해 시간이 단축된다. 그러나 맵 태스크의 결과물인 중간 단계의 데이터는 단일 리듀스 태스크에서 처리됨으로써 시간 지연이 발생한다. 따라서 본 논문에서는 단일 리듀스 태스크 처리에서 발생하는 오버로드 및 시간 지연 문제를 해결하기 위해 적응적으로 리듀스 태스크를 할당하는 스케줄링 기법을 제안하고 실험을 통해 이 기법의 성능을 검증한다.

1. 서론

최근 스마트폰, 태블릿 PC 등의 모바일 기기들이 널리 사용됨에 따라 검색 엔진, 전자상거래, 소셜 네트워크 등의 인터넷 서비스 사용자가 증가하는 추세이다. 이 사용자들이 생성하는 데이터와 서비스를 제공하기 위해 처리되는 데이터의 양이 함께 증가함으로써 원하는 데이터를 빠르고 정확하게 처리하기 위한 방법들이 요구된다. 맵리듀스(MapReduce)[1]는 데이터 집약적인 어플리케이션을 지원하여 대량의 데이터를 처리하기 위한 병렬처리 프로그래밍 모델이다.

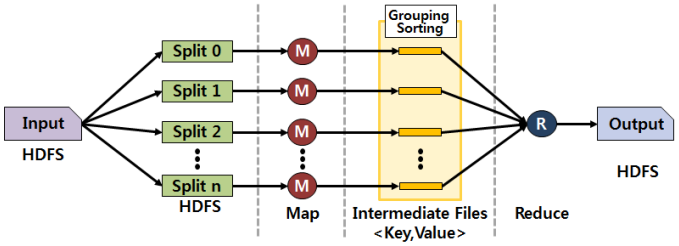
하둡(Hadoop)[2]은 맵리듀스의 오픈소스 구현으로 Yahoo!에서 개발되었으며, Yahoo!에서 뿐만 아니라 FaceBook, Amazon, Last.fm 등의 인터넷 서비스를 제공하는 회사에서 활발하게 이용되고 있다. 그러나 데이터를 맵리듀스 방식으로 처리하는 하둡 클러스터는 맵 태스크 단계 이후에 발생한 대량의 중간 단계 데이터를 하나의 리듀스 태스크에서 처리하기 때문에 리듀스 태스크 단계에서 발생하는 오버로드로 인해 수행시간이 길어진다. 이 문제점을 해결하기 위해 다양한 연구들이 수행되었다. 관련연구 [3]은 맵 태스크의 데이터 로컬리티를 향상시키기 위한 프리패칭(Prefetching) 및 리듀스 태스크의 네트워크 트래픽을 줄이기 위한 프리셔플링(Pre-shuffling) 기법을 제안하였다. 프리셔플링 기법을 통해 특정 리듀서(reducer)가 처리하는 중간 단계 데이터를 생성하는 맵 태스크의 입력 데이터를 맵 태스크에 할당하기 전에 셔플링하여 할당함으로써 네트워크 트래픽을 완화하여 리듀스 태스크의 수행시간을 감소시키고자 하였다. 관련연구 [4]는 리듀스 태스크 단계에서 리듀스 태스크 입력 데이터의 네트워크상 위치를 알 수 없기 때문에 데이터 로컬리티를 제공하지 못한다는 문제점을 지적하고 리듀스 태스크 입력 데이터의 데이터 로컬리티를 고려한 스케줄링 기법을 제안하였다. 그러나 이 연구들은 리듀스 태스크의 수행시간을 줄이고자 노력은 하였으나 네트워크 트래픽 관점으로만 접근하여 기존 하둡에서의 단일 리듀스 태스크 오버로드 문제에 대해서는 고려하지 않았다.

따라서 본 논문에서는 리듀스 태스크가 수행되는 동안 발생하는 오버로드 문제를 해결하기 위한 적응적 리듀스 태스크 스케줄링 기법을 제안한다. 맵 태스크 단계에서 발생한 중간 단계 데이터가 일정량 이상 모일 때마다 리듀스 태스크를 할당하여 맵 태스크 단계와 병렬로 처리되도록 하고, 기존의 방법에서 하나의 리듀스 태스크만 사용하여 발생한 오버로드를 여러 리듀스 태스크로 분배함으로써 전체 수행시간을 줄일 수 있게 한다. 제안하는 스케줄링 기법의 성능을 검증하기 위해 리듀스 태스크 단계의 오버로드 문제에 대해 두 가지 실험을 수행하였다.

본 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2012-0003823)

2. 단일 리듀스 태스크 처리과정

본 논문에서 제안하는 스케줄링 기법을 설명하기 전에 단일 리듀스 태스크의 오버로드 문제를 살펴보도록 한다.



(그림 1) 하둡에서의 맵리듀스 태스크 처리 과정

(그림 1)은 기존 하둡에서 맵리듀스 작업을 처리하는 과정을 도식적으로 보여준다. 맵리듀스 작업의 입력 데이터는 고정 크기의 블록으로 쪼개져서 HDFS(Hadoop Distributed File System)에 분산 저장되고, 이것은 각각 맵 태스크의 입력 데이터가 된다. 맵 태스크 단계에서 처리된 결과인 중간 단계 데이터는 키와 값의 쌍으로 구성되고 리듀스 태스크 단계의 입력 데이터가 된다. 맵 태스크 단계를 수행하는 중에 일정량의 중간 단계 데이터가 생성되면 하나의 리듀스 태스크를 할당하여 맵 태스크와 리듀스 태스크가 병렬적으로 수행되도록 한다. 그러나 다수의 맵 태스크가 처리한 중간 단계 데이터는 태스크트래커의 수에 비례하여 증가하기 때문에, 단일 리듀스 태스크의 처리속도는 중간 단계 데이터가 생성되는 속도를 따라잡지 못하고, 결국 단일 리듀스 태스크에서 오버로드가 발생하게 된다. 즉, 분산 처리된 다수의 중간 단계 데이터가 단일 리듀스 태스크로 집중되기 때문에 리듀스 단계에서 오버로드가 발생하는 것이다.

3. 적응적 리듀스 태스크 스케줄링 기법

단일 리듀스 태스크의 할당으로 인한 오버로드 문제를 해결하기 위해 적응적으로 리듀스 태스크를 할당하는 스케줄링 기법을 제안한다. 제안하는 적응적 리듀스 태스크 스케줄링의 기본 아이디어는 단일 리듀스의 오버로드를 여러 리듀스 태스크를 할당함으로써 분배시키는 것이다. 이를 위해 일정량($Threshold_{IDS}$)의 중간 단계 데이터가 쌓이면 새로운 리듀스 태스크를 할당하여 중간 단계 데이터가 분산적으로 처리될 수 있도록 한다. 새로운 리듀스 태스크가 오버로드를 충분히 분산시키게 되면 중간 단계 데이터는 $Threshold_{IDS}$ 이하로 유지된다. 새로운 리듀스 태스크의 할당은 계속적으로 발생하는 것이 아니라 리듀스 태스크의 오버로드를 분산시켜 줄 수 있을 만큼만 이루어진다. 적응적 리듀스 태스크 스케줄링은 <Algorithm 1>과 같다.

적응적 리듀스 태스크 스케줄링 기법은 기존 하둡에서의 동작과 유사하게 잡트래커와 태스크트래커 간의 HB(Heartbeat) 메시지를 이용하여 태스크트래커의 상태를 관리한다. 태스크트래커의 태스크 수행상태를 확인하여 만약 태스크트래커에 빈 슬롯이 있어 새로운 태스크의 할당이 가능하다면, 생성된 중간 단계 데이터의 크기(*IDS*; Intermediate Data Size)를 확인한다. 이것이 $Threshold_{IDS}$보다 크다면 새로운 리듀스 태스크를 할당한다. 그렇지 않은 경우 기존 맵 태스크의 할당과 동일한 스케줄링으로 새로운 맵 태스크를 할당한다.

<Algorithm 1> Adaptive Reduce Task Scheduling
Initialized as $IDS = 0$, $R = \varnothing$

```
01: When a HB message is received from node n
02:   if node n is processing a task t then
03:     record t's processing state;
04:   end if
05:   if node n completes task t then
06:     IDS = IDS + t.IDS;
07:   end if
08:   if node n has a free slot then
09:     if IDS > Threshold_IDS then
10:       assign a new reduce task r to node n;
11:       record r's information in set R;
12:     end if
13:     call a Map_task_scheduler;
14:   end if
```

여기서의 문제는 '$Threshold_{IDS}$ 값을 어떻게 설정할 것인가'이다. 이 값을 너무 작게 설정하면 할당되는 리듀스 태스크의 수가 많아지게 되어 맵 태스크를 수행할 수 있는 태스크트래커의 수가 줄어들게 되고, 분산된 리듀스 태스크의 결과를 합치는데 오버헤드가 발생하게 된다. 따라서 적절한 $Threshold_{IDS}$ 값을 설정하기 위해서는 추후 연구를 수행할 필요가 있다.

4. 실험

제안하는 적응적 리듀스 태스크 스케줄링 기법의 성능을 확인하기 위하여 단일 리듀스 태스크의 오버로드 문제를 확인할 수 있는 실험 및 리듀서의 수를 변경하면서 맵리듀스 작업의 수행시간 변화를 확인할 수 있는 실험을 각각 수행하였다.

4.1. 실험환경

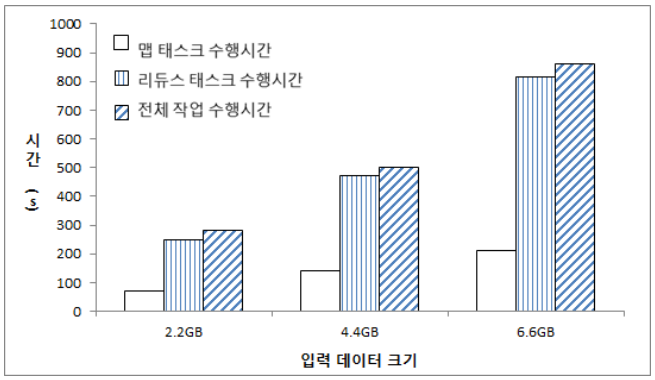
실험을 위해 마스터 노드 1대와 슬레이브 노드 4대로 클러스터를 구성하고, 각 노드는 동일하게 설정하였다. 노드의 정보는 <표 1>과 같다.

<표 1> 클러스터를 구성하는 노드의 구성

항목	모델
CPU	Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz, quad core
RAM	8GB
HDD	SAMSUNG HD502HM 500GB
Network	RTL8111/8168B PCI Express Gigabit Ethernet controller

4.2. 단일 리듀스 태스크의 오버로드

단일 리듀스 태스크의 오버로드 문제를 확인하기 위해 하둡 클러스터를 사용하여 각각 2.2GB, 4.4GB, 6.6GB의 로그 파일을 wordcount 프로그램으로 처리하였을 때의 맵 태스크 및 리듀스 태스크 수행시간, 그리고 전체 작업의 수행시간을 측정하였다. 실험 결과는 (그림 2)와 같다.



(그림 2) 입력 데이터 크기에 따른 맵리듀스 수행시간

(그림 2)에서 주의해야 할 점은 맵 태스크의 수행시간과 리듀스 태스크의 수행시간의 합이 전체 작업의 수행시간이 아니라는 것이다. 이것은 맵 태스크 단계가 완료된 뒤에 리듀스 태스크가 시작되는 것이 아니라 일정량의 중간 단계 데이터가 생성되었을 때 리듀스 태스크가 시작되기 때문이다. 맵 태스크와 리듀스 태스크의 수행시간이 오버랩 되는 부분을 제외하더라도 전체 작업 수행시간의 대부분이 리듀스 태스크의 수행시간임을 알 수 있다.

<표 2> 입력 데이터 크기에 따른 수행시간 비율

비율 \ 크기	2.2GB	4.4GB	6.6GB
맵 태스크 수행시간 비율	25%	28%	24%
리듀스 태스크 수행시간 비율	89%	94%	95%

<표 2>는 (그림 2)의 데이터를 통해 구한 전체 작업 수행시간 중 맵 태스크 및 리듀스 태스크 수행시간이 차지하는 비율을 나타낸다. 입력 데이터가 클수록 리듀스 태스크의 수행시간이 더 길어짐을 알 수 있다. 이것은 단일 리듀스 태스크의 오버로드 문제를 보여주는 실증적인 예가 된다. 실제 맵리듀스로 작업을 처리하는 시스템에서는 실험에서 사용된 데이터보다 훨씬 많은 양의 데이터를 처리하기 때문에 단일 리듀스 태스크의 오버로드 문제가 더 심각해지게 된다.

4.3. 리듀서의 수와 작업시간

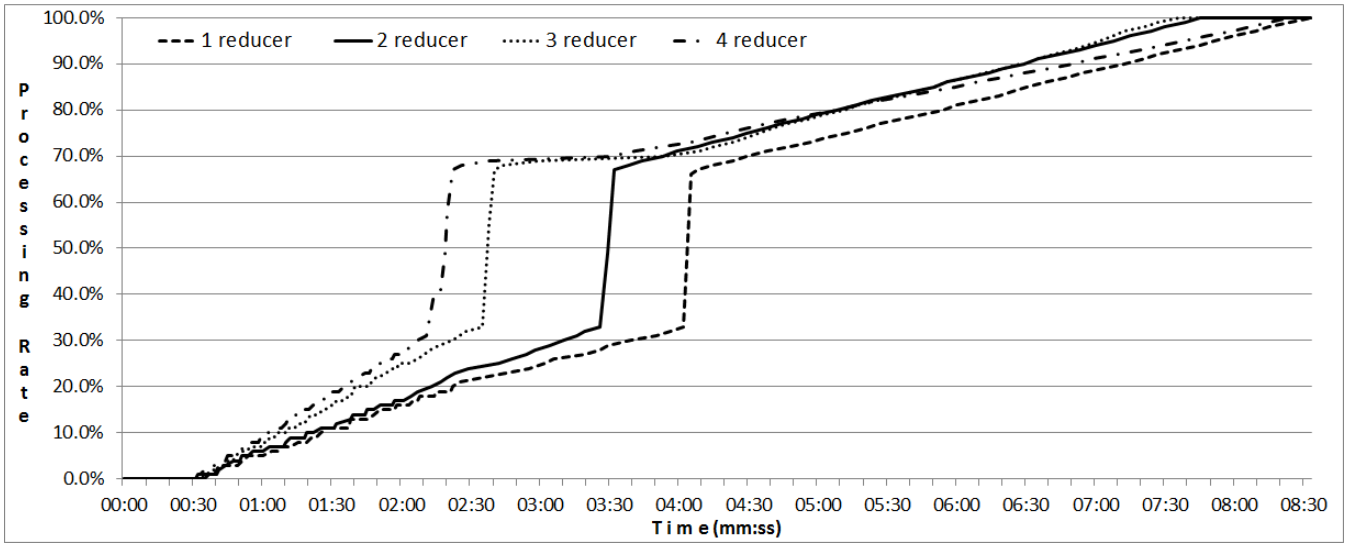
리듀서의 수가 작업시간에 미치는 영향을 파악하기 위해 리듀서의 수를 1개에서 4개까지 변화시키면서 2.2GB의 로그 파일을 wordcount 프로그램으로 처리하였을 때의 리듀스 태스크 수행시간을 측정하였다.

(그림 3)은 리듀서의 수를 변경하면서 경과시간에 따른 작업 진행률을 나타내었다. 그래프에서 특이한 점은 리듀서의 수에 상관없이 짧은 시간 동안 33~66%의 작업이 진행된다는 점이다. 이 원인을 찾기 위해 로그 파일을 분석하였고, 리듀스 태스크의 연산인 copy, sort, reduce가 순서대로 진행됨을 확인하였다. 따라서 각 연산이 리듀스 태스크 진행률 중 33.33%를 차지하므로 그래프에서 처음으로 완만하게 진행되는 단계는 copy 연산이, 급격하게 진행되는 단계는 sort 연산이, 마지막으로 완만하게 진행되는 단계는 reduce 연산이 수행됨을 의미한다. 즉, sort 연산이 copy나 reduce 연산에 비해 매우 빠르게 수행되는 연산이라는 것이다.

적응적 리듀스 태스크 스케줄링의 타당성을 확보하기 위해 다수의 리듀서를 둠으로써 얻을 수 있는 작업시간의 감소 및 결과 합병으로 인한 작업시간의 증가를 확인하고자 하였다.

리듀서의 수가 많을수록 리듀스 태스크가 병렬 수행되므로 작업시간이 감소할 것이라고 예상하였다. 이것은 작업 진행률이 0에서 66%까지 진행되었을 때의 결과를 통해 확인할 수 있다. 66%까지는 4개의 리듀서를 두고 수행하였을 때 가장 빠르게 진행되었고, 1개의 리듀서를 두고 수행하였을 때 가장 느리게 진행되었다.

또한 병렬로 리듀스 태스크가 수행되기 때문에 각각 발생한 결과를 합병하는 단계가 추가적으로 필요하기 때문에, 이로 인한 작업시간의 증가를 예상하였다. 이것은 작업 진행률이 66에서 100%까지 진행되었을 때의 결과를 통해 확인할 수 있다. 실험 결과, 3개의 리듀서를 두고 수행하였을 때 가장 빠르게 완료되었고, 1개의 리듀서를 두고 수행하였을 때 가능 느리게 완료되었다. 4개의 리듀서를 두고 수행한 경우는 copy와 sort 연산까지는 가장 빨리 끝났으나, 각 결과를 합병하는데 발생한 오버헤드 때문인지 두 번째로 늦게 완료된 것을 볼 수 있었다. 이 결과를 통해 리듀서의 개수가 많다고 반드시 작업시간이 감소하는 것은 아님을 확인할 수 있었다.



(그림 3) 리듀서 개수에 따른 리듀스 태스크 수행시간

5. 결론 및 향후연구

본 논문에서는 단일 리듀스 태스크에서 오버로드로 인해 맵리듀스 작업의 성능이 저하되는 문제를 해결하기 위한 적응적 리듀스 태스크 스케줄링 기법을 제안하였다. 중간 단계 데이터가 $Threshold_{IDS}$ 이상 생성될 때마다 새로운 리듀스 태스크를 할당하여 리듀스 태스크 단계를 병렬처리함으로써 단일 리듀스 태스크의 오버로드를 해결하고자 하였다. 이를 증명하기 위해 입력 데이터 크기에 따른 단일 리듀스 태스크의 오버로드 문제를 실험하고, 또한 리듀서의 개수를 변화시키면서 작업시간의 변화를 관찰하였다.

향후 제안한 적응적 리듀스 태스크 스케줄링을 구현하여 실제 시스템에 적용해 보고 최적의 $Threshold_{IDS}$ 값을 찾기 위한 연구 뿐 아니라 새로운 리듀스를 할당할 때 리듀스 태스크의 로컬리티도 함께 고려함으로써 네트워크 지연으로 발생하는 오버헤드를 최소화하기 위한 연구를 수행할 것이다.

참고문헌

[1] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” Commun. ACM, Vol. 51, pp. 107 - 113, January 2008.

[2] T. White, “Hadoop: The Definitive Guide”, O′Reilly Media, Yahoo! Press, 2009.

[3] Sangwon Seo, Ingook Jang, Kyungchang Woo, Inkyo Kim, Jin-Soo Kim, Seungryoul Maeng, "HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment", IEEE International Conference on Cluster Computing and Workshops 2009 (CLUSTER '09), pp.1-8, 2009.

[4] M. Hammoud, M.F. Sakr, "Locality-Aware Reduce Task Scheduling for MapReduce", IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom 2011), pp.570-576, 2011.

[5] Apache Hadoop. http://hadoop.apache.org/