# 실시간 모바일 클라우드 컴퓨팅을 위한 맵리듀스 응용 처리 기법 분석

김희재, 윤찬현
한국과학기술원 전기및전자공학과
e-mail : {kim881019, chyoun}@kaist.ac.kr

# An analysis of MapReduce application processing schemes for real-time mobile cloud computing

Heejae Kim, Chan-Hyun Youn
Dept. of Electrical Engineering, Korea Advanced Institute of Science and Technology

**요 약**

본 논문에서는 실시간 모바일 클라우드 컴퓨팅(mobile cloud computing)을 위한 맵리듀스(MapReduce) 응용 처리 기법으로써 데이터 전송 경로 관리, 노드(node) 간 다른 처리 속도로 인한 문제점 개선을 통한 성능 향상 기법들과 맵리듀스 작업의 효과적인 반복적 및 스트리밍(streaming) 실행 기법들을 분석한다.

## 1. 서론

최근 모바일(mobile) 컴퓨팅의 급격한 발전 및 해당 시장의 증가로 모바일 응용 또한 점점 복잡해지고 있다. 하지만 해당 응용들을 실행하기에 현재 모바일 기기는 하드웨어 및 배터리 한계, 모바일 기기의 이동성, 보안 문제 등의 제약조건을 가지고 있다 [1]. 따라서 이를 해결하기 위하여 모바일 클라우드 컴퓨팅(mobile cloud computing) 등의 연구가 활발히 진행되고 있다. 일반적으로 모바일 클라우드 컴퓨팅에서는 응용들의 실행 및 데이터 저장 등을 모바일 기기 외부의 클라우드에서 수행하도록 하며 이는 해당 모바일 기기 사용자들에게 배터리 수명의 증가, 데이터 저장 용량 및 처리 속도 향상, 신뢰도 증가 등의 장점들을 제공한다 [2].

본 논문에서는 특히 맵리듀스(MapReduce) [3] 응용을 실시간으로 처리하기 위한 모바일 클라우드 컴퓨팅 기술을 다루며 관련 기법들을 소개한다. 즉, 실시간 모바일 클라우드 컴퓨팅을 위한 맵리듀스 응용 처리 기법으로써 데이터 전송 경로 관리, 노드(node) 간 다른 처리 속도로 인한 문제점 개선을 통한 성능 향상 기법들과 맵리듀스 작업의 효과적인 반복적 및 스트리밍(streaming) 실행 기법들을 분석한다.

## 2. 맵리듀스 개요 및 특징 [3, 4]

맵리듀스는 거대한 양의 데이터를 효과적으로 병렬 및 분산 처리하기 위한 프로그래밍 모델로써 해당 응용은 맵(Map) 작업과 리듀스(Reduce) 작업으로 구성되어 있다. 맵 작업은 키/값(key/value) 쌍들로 구성된 입력 데이터를 처리하여 중간 데이터로써의 키/값 쌍들을 생성하는 역할을 하며 리듀스 작업은 해당 중간 데이터의 값들을 각각의 키 별로 합치는 역할을 한다. 그림 1 은 맵리듀스 응용 처리 세부 단계를 나타낸다.

- 입력 데이터를 읽고 분할하는 단계: 입력 데이터를 분할하고 해당 응용의 복사본을 클러스터 안에 배치하여 맵리듀스 응용이 병렬 및 분산 처리 될 수 있도록 한다. 해당 복사본 중 하나는 마스터로써 해당 응용의 전체 실행을 관리하며 나머지는 워커(worker)로써 맵과 리듀스 작업 등을 수행한다.

- 분할된 입력데이터를 할당해 맵 작업을 수행한 후 그 결과인 중간 데이터를 통합 및 분할하는 단계: 맵 작업이 마스터에 의하여 워커들에게 할당되며 해당 워커들은 분할된 입력 데이터를 이용하여 맵 작업을 수행한다. 여기서 맵 작업의 결과인 중간 데이터는 통합 및 분할되어 디스크에 저장된다.

- 통합 및 분할된 중간 데이터를 셔플(shuffle) 하는 단계: 각 중간 데이터의 위치가 마스터에게 전달되며 마스터는 이를 이용하여 리듀스 작업을 수행할 워커들을 결정한다. 해당 워커들은 마스터로부터 중간 데이터의 위치를 전달받는다.

- 셔플된 중간 데이터를 이용하여 리듀스 작업을 수행하는 단계: 리듀스 워커들은 중간 데이터를 읽어와서 키에 따라 그룹화되도록 이를 정렬한 뒤 리듀스 작업을 수행한다.

- 출력 데이터를 생성하고 맵리듀스 처리를 종료하는 단계: 리듀스 작업을 통하여 결과 데이터가 생성되고 마스터가 맵리듀스 작업을 종료한다.

## 3. 맵리듀스 응용 처리 성능 향상 기법

### 3.1 데이터 전송 경로 관리를 통한 성능 향상 기법 [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

본 절에서는 필요한 데이터를 얻는 시간을 줄이기 위한 방법으로써 효과적인 데이터 전송 관리를 다루며 관련 기법들은 다음과 같다.

첫째는 맵리듀스 작업에서의 입력 및 중간 데이터 로딩(loading) 시간을 줄이는 것이다. 맵리듀스 응용 처리의 각 단계에서 해당 데이터의 로딩의 경우 I/O 작업이 주를 이루고 컴퓨팅의 경우 CPU 및 메모리 등을 사용하는 작업이 주를 이루므로 이들을 순차적으로 수행하는 것은 자원 사용의 비효율성을 유발하며 결과적으로 실행 시간의 증가로 이어진다. 데이터 프리페칭은 이를 해결하기 위한 방법으로써 블록 내 프리페칭(intra-block prefetching)과 블록 간 프리페칭(inter-block prefetching)으로 구분된다. 블록 내 프리페칭은 하나의 데이터 블록을 처리할 때 컴퓨팅과 동시에 필요한 데이터를 해당 데이터 블록 안에서 프리페칭하는 것이며 블록 간 프리페칭은 각 단계에서 필요한 데이터가 해당 랙(rack)에 존재하지 않을 때 해당 데이터가 존재하는 다른 랙의 데이터를 미리 파이프라인(pipeline) 방식으로 복사해 오는 것이다. 따라서 이를 이용하여 데이터 로딩과 컴퓨팅을 병렬화 하여 맵리듀스 작업에서의 데이터 지역성을 향상시킬 수 있으며 이는 곧 성능 향상으로 이어진다. 또한 디스크에 저장되어 있는 파일에서 실행에 필요한 부분만을 저장하거나 효과적으로 데이터를 압축하여 저장하는 등의 최적화 방법 및 입력데이터의 경우 필터링(filtering)을 미리 수행하여 맵리듀스 작업에서의 부하를 줄이는 방법 또한 성능 향상에 효과적으로 이용될 수 있다.

둘째는 중간 데이터의 셔플을 효과적으로 수행하기 위한 방법으로써 셔플의 시작 시점을 적응적으로 조절하거나 프리셔플(preshuffle)을 수행 하는 것이다. 셔플의 시작 시점을 적응적으로 조절하는 것은 맵 작업을 아직 마치지 않은 상태에서도 가능할 경우 셔플을 진행하여 해당 중간 데이터를 리듀스 작업에 할당하는 것으로써 이를 통하여 셔플로 인한 시간을 감소시킬 수 있다. 하지만 해당 방법은 맵 작업을 오랫동안 수행하는 워커에 리듀스 작업이 할당될 가능성을 낮게 만들어 리듀스 작업에서의 데이터 지역성을 감소시키며 노드 간 통신으로 인한 네트워크 오버헤드(overhead)의 증가를 유발할 수 있다. 따라서 셔플의 시작 시점을 효과적으로 설정하여 맵리듀스 전체 처리 시간을 감소시키고 네트워크 오버헤드를 모두 고려할 수 있도록 하여야 한다. 또한 프리셔플은 맵 작업을 시작하기 전에 입력 데이터 등을 분석하고 각 리듀스 워커로 할당될 데이터를 미리 예측하여 해당 데이터에 대한 각각의 맵 작업을 해당 리듀스 워커 근처로 할당하는 방법으로써 이를 통하여 셔플되는 작업의 수를 줄일 수 있다.

셋째는 맵과 리듀스 작업을 합쳐 하나의 작업으로 구성하여 처리하는 방법이다. 해당 방법은 맵과 리듀스 작업 사이의 셔플 및 정렬을 건너뛸 수 있도록 하므로 실행 시간을 감소시키고 중간 데이터를 저장하기 위한 메모리 및 디스크를 절약할 수 있도록 한다.

넷째는 작업의 진행 상황 및 데이터를 관리할 수 있는 고성능의 큐 등을 이용하는 방법이다. 그 예로써 아마존의 Simple Queue Service (SQS)는 유닉스(UNIX)에서의 파이프(pipe)와 유사하게 프로세스들이 메시지를 한 쪽 끝에서 쓰고 다른 프로세스들이 메시지를 다른 쪽 끝에서 읽는 구조를 가지며 네트워크를 통한 접근이 가능하다. 이를 이용할 경우 해당 SQS를 통하여 거대한 양의 데이터를 관리할 수 있기 때문에 셔플 및 정렬과 디스크 페이징(paging)과 같은 추가적인 단계 등을 건너뛸 수 있도록 하며 중간 데이터를 디스크에 저장한 후 리듀스 워커에게 전달하는 대신 해당 SQS를 이용하여 전달함으로써 데이터 로딩 속도를 증가시킬 수 있다. 더불어 분산 메모리 캐시 등의 기술 또한 맵리듀스 작업에서의 데이터 이동을 효율적으로 수행할 수 있도록 한다.

다섯째는 비동기적 데이터 처리를 통하여 맵리듀스 작업을 수행하는 방법이다. 맵 작업과 리듀스 작업을 순차적으로 하는 것은 노드들의 처리 속도에 따라 워커들이 유휴 상태로 대기하는 경우가 생길 수 있으며 이는 자원 사용의 비효율성을 유발한다. 따라서 이는 맵과 리듀스 작업을 비동기적으로 수행함으로써 해결할 수 있으며 그 방법으로는 계층적 리듀스와 증가적 리듀스가 있다. 계층적 리듀스는 특정 개수의 맵 작업이 완료되는 시점에서 하나의 리듀스 작업을 시작하고 이를 반복하여 부분적 리듀스 작업의 결과를 생성하여 이를 트리(tree)를 이용하여 취합하는 방법이다. 이를 이용할 경우 추가적인 네트워크 오버헤드를 발생시킬 수 있지만 하나의 키를 처리하는 리듀스 작업을 병렬화 할 수 있으므로 리듀스 작업이 많은 맵리듀스 응용에 적합하다. 또한 증가적 리듀스는 필요한 리듀스 작업을 미리 생성하여 특정 개수의 맵 작업이 완료되었을 때 해당 중간 데이터를 이용하여 리듀스 작업을 수행하며 이를 모든 맵 작업이 완료될 때까지 진행하는 방법이다. 이는 맵과 리듀스 작업이 효과적으로 오버랩(overlap)되도록 하여 실행 시간을 감소시킨다.

### 3.2 노드 간 다른 처리 속도로 인한 문제점 개선을 통한 성능 향상 기법 [16, 17, 18, 19]

맵리듀스 작업에서는 맵 작업이 끝난 후 리듀스 작업이 이루어지며 그 동안 많은 중간 단계를 거치기 때문에 각 작업 간의 의존성이 존재한다. 따라서 전 단계에서의 작업이 느려지거나 데이터가 사라질 경우 이는 다음 단계에 영향을 주게 된다. 맵리듀스의 각 단계에서의 작업은 크게 디스크 안에서 혹은 네트워크를 통하여 데이터를 읽고 쓰는 작업과 컴퓨팅을 수행하는 작업으로 분류되며 여기서의 성능 저하 원인은 자원 경쟁, 네트워크 링크(link) 용량의 다양성, 데이터의 불가용성, 불공평한 작업 분할로 구분된다.

- 자원 경쟁으로 인한 성능 저하 및 해결 방안

자원 경쟁은 각 노드에서 여러 쓰레드들이 동시에 노드 안의 공유 자원을 사용하려고 할 때 발생하며 이는 성능 저하의 원인이 된다. 특히 클라우드와 같은 가상화 환경에서 하나의 물리적 머신(physical machine)에 다수의 가상 머신(virtual machine)을 구동할 때 해당 가상 머신 간의 자원 경쟁을 일으며 성능 고립을 어렵게 한다. 이를 해결하기 위한 방법으로는 성능 저하를 일으키는 작업을 중단시킨 후 다른 노드에서 재시작하는 방법, 해당 작업을 다른 노드로 복제하여 실행하는 방법, 유휴 자원으로 작업을 재분배하는 방법 등이 있다.

- 네트워크 링크 용량의 다양성으로 인한 성능 저하 및 해결 방안

맵리듀스의 각 단계에서 네트워크를 통하여 데이터를 읽고 쓸 때 한정된 네트워크 링크의 용량은 자원 경쟁을 유발하며 이는 용량이 작은 링크 안에서 특히 문제가 될 수 있다. 따라서 이를 해결하기 위하여 네트워크 인식 자원 배치가 필요하며 이를 위하여 리듀스 작업을 효과적으로 분배하여 노드 및 랙 간 거리를 가깝게 하는 등의 방법이 사용될 수 있다.

- 데이터의 불가용성으로 인한 성능 저하 및 해결 방안

맵리듀스 작업에서 작업 간의 의존성으로 인하여 데이터가 손실되거나 수행이 잘못되어 해당 작업이 재수행 되어야 할 경우 성능 저하가 생길 수 있다. 따라서 손실되거나 재수행이 필요할 가능성이 높은 데이터들을 복제하여 데이터의 불가용성을 없애는 것이 필요하다.

- 불공평한 작업 분할로 인한 성능 저하 및 해결 방안

맵과 리듀스 작업은 일반적으로 여러 노드에 분산되어 처리된다. 여기서 분배된 작업 크기 및 해당 노드들의 이종성은 각 노드에서의 실행 시간에 영향을 미치며 이러한 관점에서 작업 분할이 불공평하게 수행된 경우 특정 노드에서의 실행 시간이 길어지게 되므로 이는 성능 저하를 유발한다. 따라서 워크로드(workload)가 큰 작업을 먼저 시작하거나 적응적으로 작업을 분할하여 워크로드를 공평하게 분배하는 것이 필요하다.

## 4. 맵리듀스 작업의 효과적인 반복적 및 스트리밍 실행 기법 [20, 21, 22, 23, 24, 25, 26, 27, 28]

동일한 맵리듀스 작업이 반복적으로 실행되는 경우 맵리듀스 작업의 각 단계를 매번 수행하는 것은 자원 사용의 비효율성을 유발하고 이는 성능에 영향을 미친다. 따라서 맵리듀스 응용의 효과적인 반복적 실행을 위해서는 가능한 경우 해당 단계들을 건너뛰거나 통합하는 것이 필요하며 다음과 같은 방법들이 사용된다. 첫째는 각 실행에서의 결과를 캐싱(caching) 하는 것이다. 각 실행에서의 결과 데이터는 전체 실행의 관점에서 고정적 또는 가변적 중 하나의 성질을 갖게 되고 여기서 고정적 성질을 갖는 데이터들을 캐싱하여 다음 실행에 사용하는 것은 그 데이터를 생성하기 위한 자원 사용을 절약하도록 한다. 둘째는 각각의 맵리듀스 작업을 하나의 커다란 맵리듀스 작업으로 구성하는 것으로써 이는 각각의 실행에서의 고정적 데이터의 재로딩을 없애 성능 향상에 도움을 줄 수 있다. 셋째는 각 실행에서 리듀스 작업의 결과 데이터를 캐싱하여 고정값 및 수렴도 체크를 하는 것으로써 이는 반복적인 맵리듀스 응용의 실행에서 언제 실행을 끝낼 것인지 결정하기 위하여 복잡한 추가적인 단계가 필요하지 않도록 한다.

한편 맵리듀스 작업의 실행 요청이 스트리밍 형태로 일어날 경우 데이터 전송 방법을 고도화 함으로써 성능을 향상시킬 수 있다. 해당 데이터 전송 방법 중 하나는 파이프라인을 이용하는 것으로써 이는 각 맵리듀스 작업의 실행에서 다음 실행으로 또는 해당 작업에서의 각 단계에서 다음 단계로 데이터를 연속적으로 전달할 수 있으므로 효율적인 자원 사용을 통하여 처리 시간을 감소시키고 중간 및 결과 데이터에 대한 예측을 실행 도중에 할 수 있으므로 이를 실행에 적응적으로 반영하는 등의 장점을 가진다. 또한 데이터가 연속적으로 입력되는 경우 데이터가 입력될 때마다 해당 윈도우(window) 안에서 맵리듀스 작업을 매번 실행하는 것은 비효율적이다. 따라서 맵리듀스 작업을 연속적으로 입력되는 데이터에 따라서 단순히 업데이트가 되도록 할 경우 각 단계에서의 재수행을 줄일 수 있어 성능을 향상시킬 수 있다. 여기서 윈도우 크기 및 구두점 위치 등은 성능에 영향을 미치는 요소로써 신중하게 결정되어야 한다.

## 5. 결론

본 논문에서는 실시간 모바일 클라우드 컴퓨팅을 위한 맵리듀스 응용 처리 기법을 다루었다. 모바일 클라우드 컴퓨팅 환경에서 맵리듀스 응용을 실시간으로 처리하기 위한 기법으로써 데이터 전송 관리 및 노드 간 다른 처리 속도로 인한 문제점 개선을 통한 맵리듀스 성능 향상 기법들을 분류하고 분석하였으며 맵리듀스 작업 실행이 반복적 및 스트리밍 형태로 수행될 경우의 효과적인 실행 기법들을 소개하였다. 여기서 맵리듀스 응용 및 해당 응용을 실행하기 위한 인프라스트럭처는 각각의 구성 방식 및 그 특성에 따라 다양하게 나타나므로 효과적인 실행을 위해서는 이에 따른 데이터 전송 관리 기법 및 스케줄링 기법들이 적용되어야 한다. 또한 실시간 모바일 클라우드 컴퓨팅의 경우 응용들이 반복적으로 혹은 스트리밍 형태로 실행되는 경우가 많으므로 이를 고려하여 맵리듀스 응용을 구성하고 실행 시 중복되는 단계 등을 피하는 등의 기법들의 적용하는 것은 성능 향상 및 실시간성 보장에 도움을 줄 수 있다.

**Acknowledgement**

본 연구는 미래창조과학부 및 정보통신산업진흥원의 대학 IT 연구센터 육성지원 사업의 연구결과로 수행되었음 (NIPA-2014(H0301-14-1020)).

## 참고문헌

[1] M. Satyanarayanan, “Fundamental Challenges in Mobile Computing,” *ACM Symposium on Principles of Distributed Computing*, 1996.

[2] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, “A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches,” *Wireless Communications and Mobile Computing*, vol. 13, no. 18, 2013.

[3] J. Dean, and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *USENIX Symposium on Operating Systems Design and Implementation*, 2004.

[4] D. –S. Park, Y. –S. Moon, Y. –H. Park, C. –H. Youn, Y. –S. Jeong, and H. –S. Jang, Big Data Computing Technology, HANBIT Academy, 2014.

[5] D. Abadi, S. Madden, and M. Ferreira, “Integrating Compression and Execution in Column-Oriented Database Systems,” *ACM SIGMOD International Conference on Management of Data*, 2006.

[6] M. Elteir, H. Lin, and W. Feng, “Enhancing MapReduce via Asynchronous Data Processing,” *IEEE International Conference on Parallel and Distributed Systems*, 2010.

[7] A. Guttman, “A Dynamic Index Structure for Spatial Searching,” *ACM SIGMOD International Conference on Management of Data*, 1984.

[8] M. Hammoud, and M. F. Sakr, “locality aware reduce task scheduling for mapreduce,” *IEEE International Conference on Cloud Computing Technology and Science*, 2011.

[9] E. Jahani, M. J. Cafarella, and C. Re, “Automatic Optimization for MapReduce Programs,” *VLDB Endowment*, vol. 4, no. 6, 2011.

[10] W. Jiang, and G. Agrawal, “Ex-Mate: Data-Intensive computing with Large Reduction Objects and Its Application to Graph Mining,” *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2011.

[11] W. Jiang, V. T. Ravi, and G. Agrawal, “a map-reduce system with an alternate API for multi core environments,” *IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010.

[12] H. Liu, and D. Orban, “Cloud MapReduce: a MapReduce Implementation on top of a Cloud Operating System,” *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2011.

[13] S. Seo, I. Jang, K. Woo, I. Kim, J. -S. Kim, and S. Maeng, “HPMR: Prefetching and Pre-Shuffling in Shared MapReduce Computation Environment,” *IEEE International Conference on Cluster Computing and Workshops*, 2009.

[14] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O’Neil, P. O’Neil, A. Rasin, N. Tran, and S. Zdonik, “C-store: A Column-Oriented DBMS,” *International Conference on Very Large Data Bases*, 2005.

[15] S. Zhang, J. Han, Z. Liu, K. Wang, and S. Feng, “Accelerating MapReduce with Distributed Memory Cache,” *IEEE International Conference on Parallel and Distributed Systems*, 2009.

[16] G. Ananthanarayana, S. Kandula, A. Greenberg, I. Stoica, Y, Lu, B. Saha, and E. Harris, “Reining in the Outliers in Map-Reduce Clusters using Mantri,” *USENIX Symposium on Operating Systems Design and Implementation*, 2010.

[17] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, “SkewTune: Mitigating Skew in MapReduce Applications,” *ACM SIGMOD International Conference on Management of Data*, 2012.

[18] H. Lin, X. Ma, J. Archuleta, W. Feng, M. Gardner, and Z. Zhang, “MOON: MapReduce On Opportunistic eNvironments,” *ACM Interantional Symposium on High Performance Distributed Computing*, 2010.

[19] T. Sandholm, and K. Lai, “MapReduce Optimization Using Regulated Dynamic Prioritization,” *International Joint Conference on Measurement and Modeling of Computer Systems*, 2009.

[20] N. Backman, K. Pattabiraman, R. Fonseca, and U. Cetintemel, “C-MR: Continuously Executing MapReduce Workflows on Multi-Core Processors,” *International Workshop on MapReduce and its Applications Date*, 2012.

[21] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin, “Incoop: MapReduce for Incremental Computations,” *ACM Symposium on Cloud Computing*, 2011.

[22] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, “HaLoop: Efficient Iterative Data Processing on Large Clusters,” *VLDB Endowment*, vol. 3, no. 1-2, 2010.

[23] T. Condie, N. Conway, P. Alvaro, and J. M. Hellerstien, “MapReduce Online,” *USENIX Symposium on Networked Systems Design and Implementation*, 2010.

[24] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. H. Bae, J. Qui, and G. Fox, “Twister: a Runtime for Iterative MapReduce,” *ACM International Symposium on High Performance Distributed Computing*, 2010.

[25] J. Ekanayake, S. Pallickara, and G. Fox, “MapReduce for Data Intensive Scientific Analyses,” *IEEE International Conference on eScience*, 2008.

[26] V. Kumar, H. Andrade, B. Gedik, and K. L. Wu, “Deduce: At the Intersection of MapReduce and Streaming Processing,” *International Conference on Extending Database Technology*, 2010.

[27] D. Logothetis, and K. Yocum, “Ad-hoc Data Processing in the Cloud,” *VLDB Endowment*, vol. 1, no. 2, 2008.

[28] S. Pallickara, J. Ekanayake, and G. Fox, “Granules: A Lightweight, Streaming Runtime for Cloud Computing with Support, for Map-Reduce,” *IEEE International conference on Cluster Computing and Workshops*, 2009.