특집 06

과학 어플리케이션에 대한 MapReduce의 적용 사례 분석

목 차

1. 서 론
2. MapReduce
3. 고에너지 물리를 위한 MapReduce 기반의 데이터 분석
4. 기계 학습을 위한 MapReduce 기반의 K-means 클러스터링 알고리듬
5. 바이오 정보학을 위한 MapReduce 기반의 BLAST
6. 최적화를 위한 MapReduce 기반의 유전자 알고리듬
7. 결 론

최동훈 · 남덕윤 · 이준학
(한국과학기술정보연구원)

1. 서 론

클라우드 컴퓨팅은 주로 아마존, 세일즈포스닷컴, 구글 등과 같이 웹 기반의 as-a-Service 플랫폼으로 주목을 받아 왔지만, 대용량 데이터 처리가 필요한 기업의 정보 분석이나 과학 기술 분야의 문제 해결을 위한 경제적인 수단으로서 수요를 창출할 것으로 예상된다. 이들 문제의 특징은 대용량의 계산 자원 상에서 병렬 처리를 요구하면서, 실행 중에 자원의 소요가 지속적으로 변화한다. 따라서 계산이 끝날 때까지 최대의 소요 자원을 지속적으로 점유하는 것은 자원의 활용 측면에서 볼 때 매우 비경제적이다. 게다가, 실행하기 전에 계산에 필요한 자원의 소요량을 미리 산정하는 것조차 쉽지 않다. 이러한 이유로 클라우드 컴퓨팅에서는 사용자가 직접 쉽게 다룰 수 있는 병렬 프로그래밍 수단을 제공하고, 이에 필요한 자원을 동적으로 할당하여 데이터 분석이나 문제 해결에 효율적인 실행 시스템을 제공한다. MapReduce[1], Hadoop[2], Dryad [3]는 사용자 편의성을 고려한 병렬 프로그래밍 모형과 이에 대한 실행 시스템을 제공한다. 이중에서, MapReduce는 UC 버클리와 워싱턴대학교를 비롯하여 미국의 6개 대학교의 전산학과에서 병렬 프로그래밍 교육 환경으로 활용되고 있으며, Cloudera[4]는 일반인을 대상으로 MapReduce 교육을 제공하는 등 정보기술 시장에서 상업적인 기회를 탐색하고 있다.

본 논문에서는 과학 기술 분야의 문제 해결에 필요한 알고리듬에 MapReduce를 어떻게 적용하여 병렬 처리를 극대화하고 있는지 고에너지 물리, 기계 학습, BLAST[5], 유전자 알고리듬 등 몇 가지 적용 사례[6,7,8,9]를 중심으로 그 현황을 소개하고자 한다. 과학 기술 분야의 문제는 데이터 집약 어플리케이션과 컴퓨팅 집약 어플리케이션으로 분류할 수 있다. 컴퓨팅 집약 어플리케이션은 처리해야 할 데이터는 많지 않지만 그에 대한 계산 소요량이 대단히 많은 경우를 말하며, 처리해야 할 데이터의 용량이 대규모일 때 데이터 집약 어플리케이션이라고 부른다. 원래 MapReduce는, 크롤링한 웹 페이지의 인덱스를 효율적으로 생성, 관리 및 검색할 뿐만 아니라 시스템의 확장성(Scalability)을 통해 시스템의 유지 보수 비용을 최소화할 수 있도록, 병렬 프

로그래밍 모형 및 실행 시스템으로 구글에 의해 고안되었다. 이후 과학기술 분야에서도 자연스럽게 데이터 집약 어플리케이션에 MapReduce를 적용하는 반면, 컴퓨팅 집약 어플리케이션의 병렬화에도 적용하기 위한 많은 노력이 있었다. 이에 대한 결과로 2년 전부터 슈퍼컴퓨팅 학회나 e-Science 학회에서 성공 사례가 심심치 않게 발표되고 있다.

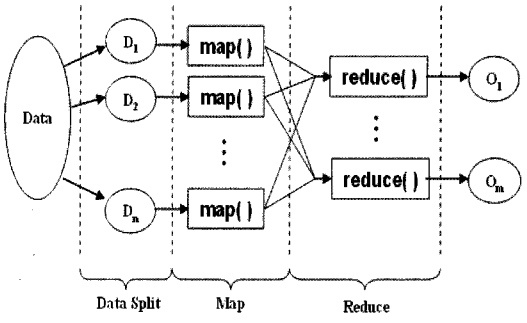
본 논문의 구성은 다음과 같다. 2장에서 MapReduce 프로그래밍 모형과 구현에 대해 간단히 설명한다. 3장부터 6장까지 과학 어플리케이션에 MapReduce 병렬 프로그래밍 모형을 어떻게 적용하여 병렬화를 극대화하였는지 관련 학회에서 발표된 논문을 중심으로 조사한 것을 소개한다. 3장에서 고에너지 물리 연구에 필요한 데이터 집약 어플리케이션을, 4장에서 기계학습에 중요한 K-means 알고리듬을, 5장에서 바이오 정보학에서 가장 많이 사용하고 있는 BLAST를, 6장에서 진화 알고리듬 중에서 대표적인 유전자 알고리듬을, 각각 소개하고 이들에 대한 MapReduce 적용 사례를 보여준다. 7장에서 이들 사례로부터 얻은 결과를 간략히 서술하고 끝맺는다.

2. MapReduce

MapReduce는 함수형 프로그래밍의 개념으로부터 고안된 병렬 프로그래밍 기술이며, 분산 컴퓨팅 환경에서의 대용량 데이터 처리를 위해 구글에 의해 제안되었다. Dean과 Ghemawat[1]는 MapReduce 병렬 프로그래밍 모형을 다음과 같이 기술하였다.

- 계산은 입력으로 key/value 쌍의 집합을 취하고, 결과로는 key/value 쌍의 집합을 생산한다. MapReduce 라이브러리의 사용자는 Map과 Reduce 함수로써 계산을 표현한다.
- 사용자에 의해 작성된 Map 함수는 입력으로 key/value 쌍을 취하고, 중간 결과로 key/value 쌍을 생성한다. MapReduce 라이브러리는 모든 중간 결과인 key/value 쌍의 집합에 대해 동일한 키 I를 갖는 key/value 쌍을 모두 찾아 하나의 그룹 I/{value1, value2,.. valuen}의 형태로 묶고, 이를 Reduce 함수에 전달한다. 이때 Reduce 함수에 전달되는 중간 결과는 I/{value1, value2,.. valuen}의 집합이 된다.
- 사용자에 의해 작성된 Reduce 함수는 여러 Map 함수로부터 I/{value1, value2,.. valuen}의 집합을 여러 받아들여, 가능한 한 가장 작은 집합으로 이들을 취합한다. 전형적으로 제로 또는 하나의 결과 값이 Reduce 수행에 의해 생산된다.

대규모로 수집된 문서에서 단어 빈출수를 세는 것은 MapReduce 기술을 설명하는 데에 사용되는 전형적인 사례이다. 데이터 집합은 작은 단위로 분리되고, map 함수는 각 데이터 단위에 대해 수행된다. map 함수는 모든 단어에 대해 〈key, value〉 쌍을 생산하며, 여기서 단어는 key이며, value는 1이다. 프레임워크는 같은 키를 가진 모든 쌍을 그룹으로 묶고, 주어진 키에 대한 value의 리스트를 reduce 함수로 전달한다. reduce 함수는 모든 값들을 취합하고, 특정 키에 대한 횟수를 생성한다. 이 경우 문서 집합에서 특정 단어의 출현 횟수이다. (그림 1)은 MapReduce 프레임워크의 데이터 흐름과 실행 단계를 보여 준다.



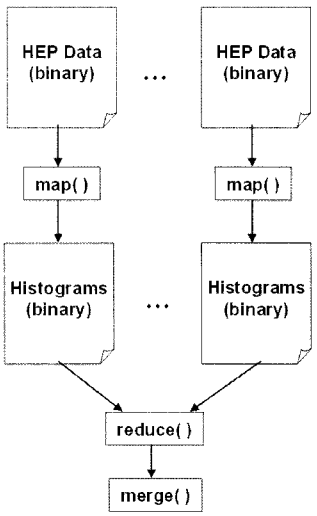
(그림 1) MapReduce 프로그래밍 모형

구글의 MapReduce 구현은 구글 파일 시스템이라 불리는 분산 파일 시스템과 연관된다. MapReduce 구현에서 중간 결과물인 〈key, value〉 쌍은 로컬 파일에 먼저 생성되고, reduce 작업에 의해 접근된다. Apache의 MapReduce 구현인 Hadoop[2]이 같은 아키텍처를 채택하고 있다. 이는 데이터뿐만 아니라 중간 결과물을 저장하기 위해 HDFS(Hadoop Distributed File System)라 불리는 분산 파일 시스템을 사용한다. HDFS는 다중 노드에 데이터를 복제해 두기 때문에, 데이터의 일부를 포함한 노드의 결함으로 인해 데이터를 활용한 계산이 멈추는 일은 없을 것이다. Hadoop은 데이터의 집약성 (locality)에 따라 MapReduce 계산을 스케쥴링하며, 이에 전체 I/O 대역폭을 절약할 수 있다.

3. 고에너지 물리를 위한 MapReduce 기반의 데이터 분석

데이터 집약 과학 분석에 있어서 대표적인 분야는 고에너지 물리[6]를 들 수 있다. 고에너지 물리 실험은 Large Hadron Collider (LHC)에서 생성하는 데이터를 처리하는 데에 초점을 두는데, LHC는 다중 레이어를 통해 여과를 한 데이터 집합에 대해 정리를 한 후에도 매년 페타바이트 이상의 데이터를 생성한다. 이렇게 생성된 데이터를 분석할 때, 확장성과 성능을 극대화하기 위해서는 병렬화 기술과 병렬화 알고리즘은 중요한 사항이다. 대부분의 병렬화 분석들은 단일 프로그램 복수 데이터(Single program multiple data, SPMD) 타입으로, 쓰레드, MPI, MapReduce과 같은 기술로 구현될 수 있다. 이러한 분석은 더 작은 계산으로 쪼개질 수 있으며, 이러한 계산들의 부분적인 결과들을 합쳐서 최종 결과를 구성하는 것은 후처리 작업에 의해 실행된다. 구현 기술의 선택은 기술 자체에서 제공되는 서비스 품질에 의존적이다. 예를 들어 하드웨어의 고장결함이 일반적인 상황에서 Hadoop과 같은 MapReduce 구현에 의해 제공되는 강건함(robustness)은 SPMD 구현에서 있어서 중요한 사항이다. 한편 MPI의 좋은 성능은 다양한 SPMD 알고리즘의 성능 측면에서는 매우 중요한 사항이다.

ROOT는 입자 물리 데이터 분석 도구들이 사용하는 프레임워크이다. 분석 함수들은 CINT라고 불리는 ROOT의 해석 언어를 이용하여 작성되어 있다. 여기서 소개하는 MapReduce 활용은 고에너지 물리 실험에 의해 생산된 데이터 파일들에 대한 분석 함수들의 모음을 실행하는 것이다. 분석 작업은 각 데이터를 처리한 후, 확인된 특징들에 대한 도표를 만드는 것이다. 도표들은 전체 분석의 최종 결과를 도출하기 위해 결합된다. 이러한 데이터 분석 작업은 데이터는 물론 컴퓨팅 집약적이며, 조합 가능한 애플리케이션 종류에 적합하다. (그림 2)는 MapReduce 구현으로 변환되었을 때, 분석에 대한 프로그램 흐름을 보여준다.



(그림 2) HEP 데이터 분석에서의 MapReduce

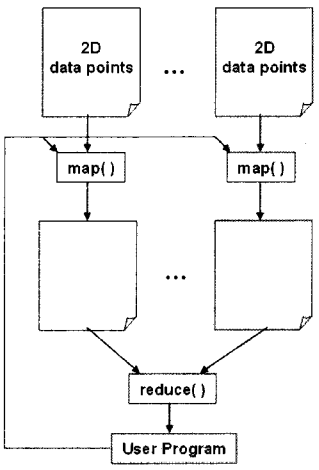
Hadoop은 Hadoop streaming이라 불리는 특별한 API를 통해 다른 언어들로 작성된 map과 reduce 함수들을 지원한다. 이 함수들은 개별 프

로세스들로 실행되며, standard output에서 함수의 결과들을 모으며, 이를 reduce 작업에 공급한다. 그러나 map 작업의 결과가 binary format이고, reduce 함수는 데이터 파일로써 결과를 기대하기 때문에, 위와 같은 방식을 사용할 수 없다. 이에 본 사례에서는 데이터 대신 도표 파일 이름을 결과로 이용하도록 map 작업에서 사용하는 분석 함수의 정의를 수정하였다. 즉, map, reduce 작업에 대해 래퍼 함수를 작성하고, CINT로 작성된 데이터 분석 함수들을 실행할 수 있도록 래퍼 함수를 사용하는 것이다. 이를 통해 Hadoop은 파일 이름을 적합한 reduce 작업에 전송한다. 데이터는 고성능 네트워크 상의 파일 시스템에 저장되고, 모든 map 작업들은 HDFS를 통해 데이터를 읽지 않아도 접근할 수 있게 된다.

map 함수에 대한 입력은 데이터 파일들의 이름들이며, 각 map 작업은 각 파일들을 처리할 것이고, 도표 파일을 생산할 것이다. map 래퍼 함수는 이 도표 파일을 읽고, HDFS를 활용하여 저장한다. map 래퍼의 결과는 HDFS의 파일 위치이다. Hadoop 실행 시스템은 이 위치들을 수집하고, 적합한 reduce 작업 즉, reduce 래퍼에 전달한다. reduce 래퍼는 HDFS로부터 도표 파일을 읽어 들이고, 이를 로컬 디스크에 복사한다. 또한 도표들의 통합을 위해 CINT로 작성된 reduce 작업을 실행시킨다. 통합된 도표는 reduce 래퍼에 의해 다시 HDFS에 저장되고 이 위치는 사용자 프로그램에 전달된다.

4. 기계 학습을 위한 MapReduce 기반의 K-means 클러스터링 알고리듬

기계 학습에서 널리 사용하고 있는 K-means 클러스터링[6]은 MapReduce 계산의 다중 반복을 수행한다. 여기서 소개하는 K-means 클러스터링의 목표는 데이터 포인트 집합을 미리 정의된 수의 클러스터로 묶는 것이다. 알고리즘의 반복은 이전 반복 계산에서 생성된 클러스터 중심점의 집합과 비교하여, 클러스터 중심점의 집합을 생성하는 것이다. 총 오차는 n번째 수행에서 생성된 클러스터 중심점과 n-1번째 수행에서 생성된 클러스터 중심 사이의 차이로, 작업은 미리 정의된 임계값으로 오차가 줄어들 때까지 계속 반복된다. (그림 3)은 K-means 알고리즘의 MapReduce 버전을 보여준다.



(그림 3) Kmeans clustering에서의 MapReduce

K-means 클러스터링에서 각 map 함수들은 데이터의 일부를 얻고, 각 수행에서 분리된 데이터를 접근한다. 데이터는 이전 수행에서 계산된 현재 클러스터 중심점들이며, map 함수에 대한 입력값으로 사용된다. Hadoop의 MapReduce API는 다중 반복에 대한 map 작업을 설정하고, 이용하는 것을 지원하지 않는다. 따라서 Hadoop 버전은 map 작업을 수행할 때마다 데이터를 읽어 들여야 한다. Hadoop은 HDFS를 통해 클러스터 중심점의 일부를 reduce 작업으로 전달한다. Reduce 작업이 클러스터 중심점을 입력 받으면, 새로운 클러스터 중심점을 계산하여 HDFS에 저장한다. 사용자 프로그램은 HDFS을 읽어 들여, 새로운 클러스터 중심점들과 이전의 클러스터 중심점들 간의 차이를 계산한다. 이 차이가 미리

지정한 임계값보다 크면, 사용자 프로그램은 새로운 클러스터 중심점들을 입력으로 하여 MapReduce의 새로운 순환을 반복적으로 수행한다.

5. 바이오 정보학을 위한 MapReduce 기반의 BLAST

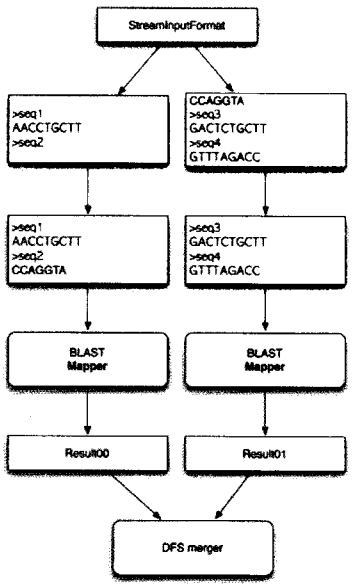
CloudBLAST[7]는 BLAST에 대한 클라우드 컴퓨팅 기술을 적용한 환경(CloudBLAST)을 구축한 것으로, BLAST는 바이오 정보학에서 가장 많이 사용하는 응용프로그램 중의 하나이다. BLAST[5]는 Basic Local Alignment Search Tool의 약자로 BLAST는 염기서열 또는 단백질 서열간의 local similarity를 찾아내는 알고리즘이다. BLAST와 같은 빠른 속도를 내는 알고리즘이 개발되기 전에는 일반적인 다이나믹 프로그래밍 기법을 이용하여 데이터베이스에서 단백질이나 염기 서열에 대한 검색을 수행하는 일은 매우 많은 시간을 요하는 작업이었으나 BLAST는 기존 다이나믹 프로그래밍 기법보다 약 50배 이상 빠른 속도를 내기 때문에 데이터베이스에 대한 서열 검색을 자주 수행하게 되는 바이오 정보학 분야에 큰 기여를 하고 있다. BLAST가 휴리스틱(heuristic) 알고리즘이기 때문에 최적의 alignment를 보장할 수는 없지만 대용량의 데이터베이스에서 관련된 비슷한 서열을 검색하는 용도로는 문제없이 작동한다. BLAST는 바이오 정보학 분야에서 다양하게 사용되고 있지만 기능적, 진화적 관계를 유추하고 유전자 family를 확인하는 용도로 사용되는 것이 일반적이다.

BLAST는 k개의 서열의 집합 S = {s1,...sk}가 주어졌을 때 각 si 서열을 n개의 서열로 이루어진 데이터베이스 D = {ds1,...,dsn}의 모든 서열과 비교를 수행하고 각 비교시 마다 통계적 유의성(statistical significance)을 계산하게 된다. 일반적으로 S는 D보다 그 크기가 매우 작다. 예를 들어 쿼리를 하고자 하는 염기서열은 약 1000개 정도일 경우 데이터베이스는 수십억의 염기서열로 이루어져있다.

병렬 버전의 BLAST도 BLAST의 수행 속도를 높이기 위해 개발되어 널리 사용되고 있다. MPI와 Pthreads를 이용하여 개발되었으며 Window, Linux, Solaris, Mac OS X, AIX등등의 다양한 플랫폼에서 실행 될 수 있도록 포팅되어 배포되고 있다. BLAST를 병렬화 하기 위해 많은 방법이 사용되고 있지만 주로 질의 분산, 해쉬 테이블 분할, 컴퓨팅 병렬화, 데이터베이스 분할 등의 방법이 사용되고 있다.

CloudBLAST는 클라우드 컴퓨팅 환경 상에 질의를 분산 수행함으로써 BLAST를 병렬화하고 성능을 개선하는 방법을 사용한다. 즉, 질의를 하고자 하는 서열의 집합 S = {S1, ... , Sk}를 i개의 부분 집합으로 나누어 i개의 서로 다른 컴퓨팅 노드 Wi에서 BLAST를 수행한다. 이러한 질의 분산을 통한 병렬화에서는 이상적인 컴퓨팅 노드의 수에 대한 추정, 로드 밸런싱, 발생 가능한 결함에 대한 회복 등에 대한 문제점이 존재한다. 이러한 문제를 효율적으로 해결하기 위하여 CloudBLAST는 (그림 4)와 같이 MapReduce를 활용한다. BLAST를 MapReduce 프레임워크를 이용하여 수행하기 위해서는 MapReduce의 각 map 함수가 서열의 부분 집합에 대해 순차적 BLAST 프로그램을 실행하도록 하는 방법을 이용할 수 있다. 기본적인 BLAST의 경우 추가적인 후처리 과정이 필요하지 않음으로 reduce는 생략할 수 있다. CloudBLAST에서는 MapReduce의 오픈 소스인 Apache Hadoop을 활용하여 NCBI의 BLAST2의 병렬화를 구현하고 있다. CloudBLAST는 Hadoop에서 제공하는 손쉽게 기존에 존재하는 애플리케이션을 수행할 수 있도록 streaming extension을 사용하여 BLAST를 각 자원에서 수행한다. 이때 Hadoop이 입력 파일을 파일 내부의 형식을 무시하고 단

순히 동일한 크기의 파일들로 분할하는 일을 방지하기 위해, 패턴 레코드 리더를 개발하여 파일 내부의 key/value 쌍이 유지 될 수 있도록 한다. 최종적으로 reduce를 사용하지 않고 HDFS에서 제공하는 파일 합병 명령을 사용하여, 각 자원에서 수행된 결과를 병합한다. CloudBLAST에서는 BLAST라는 응용 프로그램을 사용하였지만 비슷한 실행 방식을 갖고 있는 HMMER, Megablast 등의 프로그램에 MapReduce를 적용하는 것도 가능하다.



(그림 4) MapReduce를 이용한 BLAST의 병렬화

CloudBLAST는 검색의 대상이 되는 데이터베이스를 하나의 가상 이미지 템플릿으로 구성하여 관리를 하고, 이 가상 이미지 템플릿을 클라우드 인프라스트럭쳐가 지원하는 복제 기능을 이용하여 각 가상 자원으로 복제하여 BLAST에 사용하는 방식을 채택하고 있다. 이러한 방식을 채택함으로써 수시로 업데이트되는 데이터베이스의 복제에 대한 문제점을 해결하고, 각 가상 자원마다 가상 데이터 이미지를 갖게 함으로써 원격 데이터를 사용하는 것 보다는 더 뛰어난 성능을 발휘 할 뿐 아니라 복잡한 분산 데이터 관리 메커니즘을 단순화하는 효과를 가져 온다.

CloudBLAST는 Hadoop을 활용하여 병렬 BLAST 알고리듬을 실행할 수 있는 환경을 구현한다. 통상적으로 그리드와 같은 분산 환경에서의 각 컴퓨팅 자원은 운영체제와 라이브러리 등 각각 서로 다른 환경을 갖추고 있다. 하지만 특정 응용프로그램을 분산환경에서 수행하고자 하는 경우 각 컴퓨팅 자원에는 동일한 버전의 라이브러리 혹은 동일 운영체제, 혹은 추가적인 응용프로그램이나 네트워크 프로토콜 등이 필요하다. CloudBLAST에서는 이러한 문제를 해결하기 위해 가상화 기술을 사용하여, BLAST 실행 환경을 캡슐화하였다. 가상화 기술을 통해, 공유 자원을 더욱 수월하게 관리할 수 있으며, 사용자 간에 실행 환경의 분리가 가능하다. 또한 이종의 자원들 위에서 동일한 실행 환경을 사용하여 작업을 수행할 수 있는 환경을 제공할 수 있다. 또한 각기 다른 사이트에 분산되어 있는 가상 자원을 연결하기 위해 ViNe라는 가상 네트워크 기술을 이용하여 지역적으로 떨어져 있는 다수의 자원을 하나의 클라우드로 사용하도록 하였다.

6. 최적화를 위한 MapReduce 기반의 유전자 알고리듬

유전자 알고리듬은 문제 공간을 개체군으로 추상화하여, 연산의 순환을 통해 최적의 개체를 찾는다. 개체는 일련의 기호로 표현되고, 각 순환 단계는 재생, 교배(cross-over), 돌연변이(mutation), 평가, 선택연산을 통해 새로운 세대의 개체군을 생산한다. 개체군의 한 세대가 조상으로 주어졌을 때, 재생 연산은 여러 조상을 교배하여 자손을 생성하고, 돌연변이 연산은 각 자손에 간단한 확률적 변화를 수행하여 그에 대한 돌연변이 버전을 생성한다. 평가 연산은 목적 함수(objective function)에 따라 자손을 평가하고, 선택 연산은 새로 생성된 개체군에서 국지적 최

적해를 선택한다. 이 과정은 전역적 최적해를 발견할 때까지 반복 수행된다. 이들 연산 중에서 평가와 선택 연산은 대부분의 시간을 소모한다.

병렬 유전자 알고리듬[8]은 다음과 같이 주어진 개체군을 여러 개의 작은 하위 개체군으로 나누고 이들 각 하위 개체군에 worker GA를 하나씩 할당하여 병렬 처리를 수행한다. 교배와 돌연변이 연산을 수행하고 나면, 이웃하는 worker 간에 개체 집합이나 통계량을 서로 교환하는 통신 단계가 있다. 통신 단계가 끝나면 이들 자손 중에서 최적의 자손을 선택하여 다음 세대로 진화하기 시작한다. 이때, 각 개체 P는 T는 최적해가 생성되었는지 결정하는 목적 함수이다.

```
function PGA
  t = 0          /* index of generation*/
  P[0] = a1[0],···, au[0]   /* initialization */
  Evaluation(a1[0],···, au[0])
  while not T(P[t]) do
    P'[t] = Mutation (Crossover(P[t]))
    Evaluation(a'1[0], ···, a'u[0])
    〈Communication〉
    P[t+1] = Selection(P'[t])
    t = t+1
  endwhile
  eturn Optimum(P[t])
```

선택 연산은 최적의 값을 갖는 개체를 선택한다. 평가 후의 개체는 u개의 요소로 구성된다. 두 개체 Px, Py가 주어졌을 때, Px의 모든 요소가 이에 대응되는 Py의 모든 요소보다 크면 Px가 Py보다 크다고 판단한다. 이렇게 하여 최적의 값을 선택한다.

6.1 MRPGA: MapReduce를 이용한 병렬 유전자 알고리듬

MRPGA[8]에서 MapReduce는 PGA의 여러 단계 중에서 시간을 가장 많이 소모하는 부분을 병렬화하기 위해 사용된다. map 연산은 국지적 평가 단계를 표현하는 데 사용되고, 통신 단계는 reduce 연산을 위한 관련 입력 자료를 수집함으로써 완료된다. 반면에 선택 단계는 reduce 연산을 한번 수행하여 완성되는 것이 아니라, 국지적 선택과 전역적 선택을 위해 두 번의 reduce 연산을 수행하여야 전역적인 최적해를 찾을 수 있다. 따라서 PGA를 위한 MapReduce 모형은 map, reduce, reduce의 세 단계로 구성된다.

MRPGA를 위한 map과 reduce의 입력과 출력의 키/값 쌍은 다음과 같다.

map:: (key1, value1) => list (key2, value2)
reduce:: (key2, list(value2)) => list(value3)
reduce:: (key3, list(value3)) => list(value4)

입력 키/값 쌍의 데이터 유형은 다음과 같다.

Key1: integer, Value1: individual
Key2: integer, Value2: set of individuals
Key3: individual, Value3: integer

6.2 map 단계

순환의 각 단계마다 각 개체에 대해 map 연산은 한번씩 호출된다. map 함수의 입력으로서 key는 개체의 인덱스이고 value는 개체이다. map 연산은 value로부터 개체를 추출하여 평가를 수행하고, 중간 결과를 전송한다. map 단계에서 생성된 결과는 로컬 노드의 데이터베이스에 저장된다. 표준 MapReduce와 달리, 중간 결과는 키에 의해 분할되는 것이 아니라 위치에 의해 분할된다. 이러한 분할 정책은 각 reduce 연산에 필요한 데이터 수집이 원격 노드를 포함하지 않고 로컬 노드에 국한시키는 효과를 가져온다. map 연산이 동일한 노드 내에서 생성한 중간 결과는 키를 기준으로 reduce 연산 1단계에서 병합된다.

```
function map (key, value)
    P = a1[0],…, au[0]=Individual(value)
    /* translation of problem space */
    P' = Evaluation(a'1[0], …, a'u[0])    /*
    evaluation */
    Emit (default_key, P)          /* submit
    immediate results */
```

6.3 reduce 1단계

reduce 연산의 1단계는 map 연산이 생성한 각 분할에 대해 수행된다. Reduce 연산은 value_list로부터 개체군을 추출하고 이들 개체군에 대해 선택 연산을 수행하여 국지적 최적해를 선택한다. 끝으로 reduce 연산은 선택된 결과를 final_reducer에 대한 입력으로 전송한다.

중간 결과의 키는 개체이고, 값은 숫자이다. reduce 1단계에서 생성된 모든 중간 결과는 reduce 연산의 2단계에 대한 입력으로 수집된다.

```
function reduce1 (key, value_list)
  i = 0    /* index variable */
  foreach value in value_list
    P[i]=a1[i],…, au[i]=Individual(value)
    i++
  P'=Selection (P)    /* perform local
  selection */
  foreach individual in P'
    Emit (individual, 1)
```

6.4 reduce 2단계

Reduce 연산의 2단계는 전역적 선택을 위한 것이고, 각 순환의 끝에서 한번씩 호출된다. Reduce 2단계는 reduce 1단계가 생성한 중간 결과를 받아서 현재 세대에 대한 최종 선택을 수행한 결과를 생성한다.

이 결과는 MRPGA의 다음 순환에 대한 입력으로 사용된다.

Reduce 1단계에서 선택된 국지적 최적 개체는 전역적 최적 개체를 찾기 위해 병합과 정렬을 필요로 하는데, 이러한 작업은 실행 시스템에 의해 수행된다.

```
function reduce2 (key, value)
    P = a1[0],…, au[0]=Individual(key)  /*
    translation */
    P' = Selection (P)    /* perform global
    selection */
    Emit (P', 1)
```

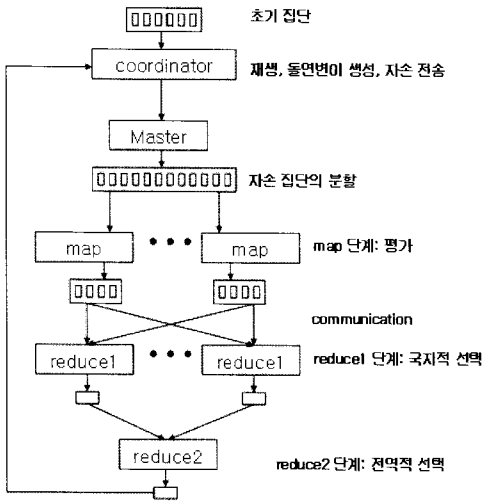
6.5 coordinator

개체군 진화를 위한 순환의 수행은 coordinator를 필요로 한다. MRPGA에서 coordinator는 재생, 돌연변이, 스케줄러에게 자손의 전송 등을 수행하고, 진화의 각 순환마다 최적의 개체를 수집한다. MRPGA에서 사용자는 분산 컴퓨팅에 대한 어려움을 직면하지 않아도 된다. 대신에 사용자는 하나의 map, 두 개의 reduce, 하나의 coordinator를 포함하는 컴포넌트를 프로그래밍하면 된다. MRPGA 실행 시스템은 map, reduce 등의 병렬 수행을 제어한다.

```
function MRPGA
    t = 0        /* index of generation*/
    P[0]= a1[0],…, au[0]   /* initialization */
    Evaluation(a1[0],…, au[0])
    while not T(P[t]) do
      P'[t] = Mutation (Crossover(P[t]))
      SendToScheduler(P'[t])
      P[t+1] = ReceiveFromScheduler (t)
      t = t+1
    endwhile
    return P[t]
```

6.6 MRPGA 실행 시스템 구조

MRPGA의 실행 시스템은 (그림 5)와 같이 하나의 master, 다수의 map worker와 reduce worker로 구성된다. Master는 병렬 작업의 실행을 스케줄링하고, map worker는 map 함수를 실행하며, reduce worker는 reduce 함수를 실행한다.



(그림 5) MRPGA 실행 시스템 구조

MRPGA 실행 시스템의 제어 흐름은 다음과 같다.

① coordinator는 자손을 생성하여 돌연변이 연산을 수행한 후에, 자손을 master에게 전송한다.

② master는 자손을 m개로 부분으로 분할하여 각각을 m 개의 map 작업으로 만든다. 이때 m은 map 작업의 병렬화를 극대화할 수 있도록 선택한다. 일반적으로 이 값은 컴퓨터의 수보다 크다.

③ 자손의 각 부분은 map worker가 있는 컴퓨터로 전송된다. map worker는 입력 부분에 속한 개체에 대해 map 함수를 반복적으로 수행한다. map 함수가 생성한 중간 결과는 국지적 저장소에 저장된다.

④ 각 reduce worker는 reduce 연산의 1단계를 수행한다. 이때, reduce worker는 보통 국지적 노드로부터 입력 데이터를 받지만, 이질적인 경우 모든 worker에게 부하를 균등하게 배분하기 위해 몇몇 reduce worker는 이웃 노드로부터 중간 결과를 받아들인다.

⑤ reduce worker는 reduce1 함수를 호출하여 국지적 최적해를 선택하여 국지적 노드에 저장한다.

⑥ reduce 2단계에서 reduce worker는 reduce2 함수를 호출하여 전역적 최적해를 최종 결과로 생산한다.

⑦ 최종 결과는 사용자에게 반환되어 MRPGA에서 다음 순환의 시작점이 되며, 이러한 순환은 설정된 요구사항을 만족하는 최적해가 나올 때까지 반복된다.

표준 MapReduce 실행 시스템과 달리, MRPGA는 전역적 최적해를 선택하기 위한 reduce 2단계가 추가되었다. 실행 중에 발생하는 결함을 단순하게 처리하기 위해 master는 각 순환에서 선택된 전역적 최적해를 복제한다. 만일 몇몇 노드가 실행 중 결함이 발생하여 사용 불가하게 되면, 바로 전의 순환 결과를 받아들여 다시 실행하면 된다. 이러한 결함 포용 방법은 표준 MapReduce 실행 시스템과 달리, 복잡한 분산 파일 시스템이 없이도 신뢰도를 제고한다.

7. 결 론

MapReduce 모형은 분산 환경에서 과학 어플리케이션에 대한 병렬 프로그램의 개발을 단순화하기 위한 병렬 디자인 패턴을 제공한다. 데이터와 컴퓨팅의 규모가 큰 과학 어플리케이션을 풀 때에는, 큰 문제 공간을 여러 개의 작은 공간으로 나누고 작은 공간에서 작업의 실행을 자동으로 병렬화하는 일이 자주 일어난다. 기존의 그리드 컴퓨팅과 같은 분산 환경에서 진행되었던 MPI 기반의 병렬화와 달리, MapReduce는 작업 실행의 자동화를 통해 사용자가 병렬 작업의 실행을 조정 통제해야 하는 어려움을 덜어준다. 분

산 환경에서 병렬 프로그램의 개발은 분산 컴포넌트 간의 통신과 동기화와 같은 어려움을 동반하고 있으며, 이질성과 잦은 장애 발생에 대한 문제 해결도 필요하다. 클라우드 컴퓨팅은 이러한 문제에 대한 해결책도 제시하고 있다. 현존하는 MPI 기반의 병렬 프로그램은 클라우드 컴퓨팅에 부적합하다. 이러한 이유로, 클라우드 컴퓨팅의 증가 추세를 고려할 때 이들 알고리듬을 클라우드 컴퓨팅 환경에서 실행할 수 있는 방법을 탐색하는 것은 매우 중요하다.

본 논문에서는 몇 가지 연구 사례의 조사 분석을 통해, 과학 어플케이션에 대한 MapReduce 기반의 병렬화 방법을 알아 보았다. 이를 통해 프로그램의 구조가 MapReduce 패러다임에 맞는 어플리케이션이라면 쉽게 map과 reduce를 정의하여 병렬화를 통한 성능의 향상 또는 분산화를 실현 할 수 있음을 알게 되었다. 데이터 집약 어플리케이션은 단 한번의 map, reduce의 실행으로 간단하게 병렬화가 가능하다. 반면, 컴퓨팅 집약 어플리케이션의 경우에는, MapReduce가 순환 구조를 제공하지 않기 때문에, 이를 위한 실행 시스템의 확장 노력이 필요함을 알게 되었다. 실제 K-means 나 MRPGA의 경우, 순환 구조의 추가를 통해 컴퓨팅 집약 어플리케이션에 대한 병렬화 방법을 제시하고 있다. MRPSO[]는 이에 대한 더욱 복잡한 경우를 보여준다. 이들 컴퓨팅 집약 어플리케이션의 사례는 순환 구조의 추가로 인해 노드 수 증가에 따른 실행 시간의 증가는 순환 구조에 의한 부하가 무시할 수 없음을 보여준다.

컴퓨팅 집약 어플리케이션의 사례로부터 알게 된 흥미로운 것은 이들에 대한 MapReduce 적용 방법이 서로 다르다는 것이다. 비록 같은 진화 알고리듬(evolutionary algorithm)이지만 MRPGA와 MRPSO[9]의 방법이 서로 다르다. 이것은 프로그래머가 MapReduce를 활용하여 병렬 프로그래밍을 하는 데에 상당히 큰 장애가 될 수 있다. 이러한 이유로 같은 부류의 알고리듬을 동일한 실행 시스템을 통해 병렬화 할 수 있는 방법을 탐색하는 것은 향후 연구 과제로 매우 의미 있고 중요하다.

참고문헌

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Communications of the ACM, 51(1):107-113, January 2008.

[2] Apache Hadoop, http://hadoop.apache.org/

[3] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," European Conference on Computer Systems, March 2007

[4] Cloudera, Hadoop training and support [online], Available from: http://www.cloudera.com/.

[5] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ, "Basic local alignment search tool," Journal of Molecular Biology 215(3):403 - 410, 1990

[6] J. Ekanayake, S. Pallickara, and G. Fox, "MapReduce for Data Intensive Scientific Analyses," 2008 Fourth IEEE International Conference on eScience, pp277-284, December 2008

[7] Andréa Matsunaga, Maurício Tsugawa, José Fortes, "CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications," 2008 Fourth IEEE

International Conference on eScience, pp222-229, 2008

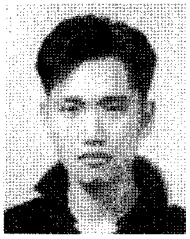
[8] C. Jin, C. Vecchiola and R. Buyya, "MRPGA: An Extension of MapReduce for Parallelizing Genetic Algorithms," 2008 Fourth IEEE International Conference on eScience, pp222-229, 2008

[9] A. McNabb, C. Monson, and K. Seppi, "Parallel PSO Using MapReduce," The Congress on Evolutionary Computation ,Singapore, 2008

저자약력

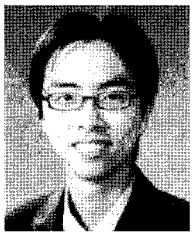
최 동 훈

1981년 2월 서울대학교 계산통계학과 졸업(학사)
1983년 2월 한국과학기술원 전산학과 졸업(석사)
1989년 6월 Northwestern University 전산학과 졸업(박사)
1983년 2월~1986년 8월 한국증권전산㈜ 과장대리
1989년 8월~1992년 2월 한국국방연구원 선임연구원
1992년 3월~1999년 2월 동덕여자대학교 부교수
2005년 2월 현재 한국과학기술정보연구원 책임연구원
관심분야 : 데이터베이스, 병렬 처리
이 메 일 : choid@kisti.re.kr



남 덕 윤

1999년 2월 포항공과대학교 컴퓨터공학과 졸업 (학사)
2001년 2월 한국정보통신대학교 공학부 졸업 (석사)
2006년 8월 한국정보통신대학교 공학부 졸업 (박사)
2004년 7월 ~현재 한국과학기술정보연구원 선임연구원
관심분야 : 분산 컴퓨팅, 미들웨어, 그리드컴퓨팅
이 메 일 : dknam@kisti.re.kr



이 준 학

2003년 2월 한국과학기술원 전산학과 졸업(학사)
2005년 2년 한국과학기술원 바이오시스템학과 졸업(석사)
2005년 2년~현재 한국과학기술정보연구원 연구원
관심분야 : 바이오인포매틱스, 그리드컴퓨팅, 워크플로우
이 메 일 : juneh@kisti.re.kr