Julia 분산-병렬 모델을 이용한 K-Means 알고리즘 최적화

김민성*O 변석우** 우균*
*부산대학교 정보융합공학과, **경성대학교 소프트웨어학과
{msungkim, woogyun}@pusan.ac.kr, swbyun@ks.ac.kr

Optimization of K-Means Algorithm Using Julia Distributed-Parallel Model

Minsung Kim*O Sugwoo Byun** Gyun Woo*
*Dep. of Information Convergence Engineering, Pusan National University,
**School of Computer Science & Engineering, Kyungsung University

요 약

Julia는 기술 컴퓨터 과학분야에서 수학, 엔지니어링, 과학 분야에 적용되는 고성능 기반의 동적 프로그래밍 언어이다. 대부분의 컴퓨터는 두 개 이상의 코어를 가지고 있으며, CPU의 멀티 코어를 활용하면 계산을 보다 신속하게 완료할 수 있다. Julia는 메모리 도메인에서 실행될 수 있도록 메시지 전달을 기반으로 한 분산 프로그래밍과 공유 메모리 기반의 병렬 프로그래밍을 지원한다. 본 논문은 Julia의 분산-병렬 프로그래밍을 이용해 K-Means알고리즘을 최적화하고, 다른 분산-병렬모델과의 성능 비교 수행을 통해 Julia의 분산-병렬 모델의 성능에 대해서 알아보고자 한다.

1. 서론

Julia는 고성능 기반의 수치 해석 및 계산 과학 분야, 다목적 프로그래밍에 효율적으로 사용되는 고수준의 동적 프로그래밍 언어이다[1]. Julia 언어는 LLVM을 기반으로 하며 Just-In-Time 컴파일을 지원한다. 또한, 디스패치, 다양한 라이브러리 지원 등이 있다.

또한, Julia는 병렬 컴퓨팅에서 경량 스레드 및 다중 스레딩 등을 지원하며, 분산 컴퓨팅에서 멀티 코어 활용 및 분산 처리를 지원한다. 특히 메시지 전달 구현의 경우는 일반적으로 단방향이므로 하나의 프로세스만 명시적으로 관리하면 되기 때문에 메시지 전달 인터페이스의 표준인 MPI와는 다르게 구현되며 함수 호출과 같은 상위 수준의 작업처럼 작업할 수 있어 효율적으로 구현할 수 있다는 장점이 존재한다. 이러한 장점을 활용해 Julia에서는 고용량의 데이터를 처리 및 분석 시에 고성능을 기반으로 신속하게 연산할 수 있다.

Julia 분산-병렬 모델의 이점을 바탕으로, 본 논문은 대표적인 데이터 군집화 알고리즘인 K-Means 알고리즘의 수행 성능을 최적화함과 동시에 분산 컴퓨팅의 표준인 MPI(Message-Passing-Interface)와 병렬 컴퓨팅의 표준인 OpenMP 분산-병렬 모델과의 성능 수행 결과를 비교 후 Julia의 분산-병렬 모델의 성능에 대해 알아보고자 한다.

*이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2014-3-00035, 매니코어 기반 초고성능 스케일러블 OS 기초연구 (차세대 OS 기초연구센터)). *교신 저자: 우균(부산대학교, woogyun@pusan.ac.kr).

2. 관련 연구

2-1. 분산-병렬 컴퓨팅

분산 컴퓨팅은 컴퓨터 과학의 한 분야로, 네트워크에 연결된 여러 컴퓨터의 처리 능력을 이용해 메시지 전달을 기반으로 연산을 수행한다. 하나의 병렬 컴퓨팅의 경우 로컬 노드의 프로세스에서 동시에 많은 계산을 수행 가능 하게 하는 연산 수행 방식을 가진다.

대표적인 분산-병렬 모델로는 일반적으로 MPI(Message-Passing-Interface)[3]와 Open MP[4]를 혼합한 하이브리드 모델[5]이 있다. 이러한 하이브리드 모델은 그림 1과 같이 동작하는 시스템을 의미한다.

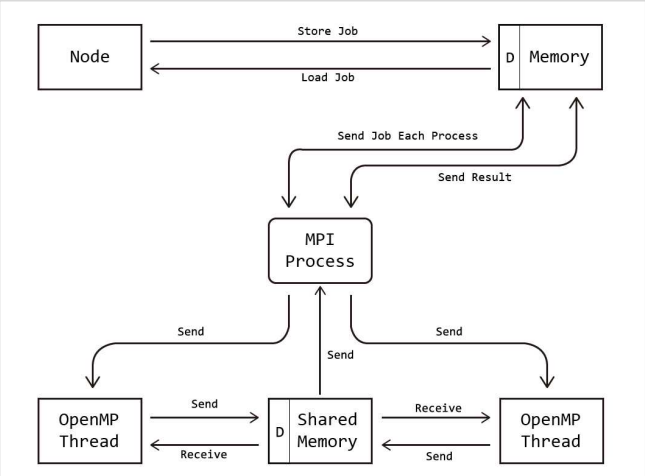

그림 1 MPI-OpenMP Hybrid 모델

그림 1과 같은 하이브리드 모델은 멀티 스레딩이 가능하므로, 다수의 스레드와 함께 작업을 수행할 수 있어 프로세스 당 작업의 효율성이 높으며, 단일 컴퓨터에 설치된 CPU의 자원만 활용할 수 있다는 한계점을 극복하는 분산 시스템과 혼합하여 다수의 노드에 작업을 분산시켜 프로그램을 실행할 수 있기 때문에 효율적이며 신속하게 연산할 수 있다.

이러한 이유로, 본 논문에서는 Julia의 분산-병렬 모델과 비교를 진행할 모델로 C 언어 기반의 MPI-OpenMP의 하이브리드 모델로 선정했다. 이유로는, C 언어는 고성능 기반의 언어임과 동시에, 메시지 전달 인터페이스의 표준인 MPI와 C, C++, Fortran 언어의 병렬 프로그래밍 표준인 OpenMP에 가장 최적화가 잘 되어있다는 이유에서 선정되었다.

2-2. K-Means 알고리즘

본 논문에서는 분산-병렬 모델 성능을 비교하기 위해 K-Means 알고리즘을 사용하고자 한다. K-Means 알고리즘은 주어진 데이터를 k개의 군집으로 묶는 알고리즘으로, 각 군집 간 거리 차 분산을 최소화하는 방식으로 동작한다. K-Means 알고리즘은 군집화 알고리즘 방법 중 분할법에 속하는 알고리즘으로, d-차원 데이터 객체 집합이 주어질 때, 데이터 객체들을 각 집합 내 객체 간 유사도를 최대로 하는 k개의 클러스터의 집합으로 분할한다[6]. 식 1과 같이,

$$\underset{S}{\arg\min} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2$$

식 1 K-Means 알고리즘의 거리계산

군집 간 중심점을 기준으로 객체 간 거리를 계산해, 군집 내 객체 간 거리의 제곱 합을 최소로 하는 군집을 찾는 것으로 수행을 종료한다.

K-Means 알고리즘의 계산 복잡도에 영향을 미치는 요소는 군집의 수인 k의 값에 따라서 영향을 받게 된다. 따라서 K-Means 알고리즘은 군집의 개수를 결정하는 k값에 따라 차이가 존재한다. 주어진 데이터 집합에서 구하고자 하는 군집의 수를 설정할 때 다양한 방법론을 사용해 구할 수 있다. 하지만, 분산-병렬 모델 간 수행 성능 비교와 수행 프로세스 수에 따른 수행 최적화를 하고자 함이 실험의 목적이기 때문에 임의의 k값과 많은 양의 데이터를 사용해 수행 성능을 평가했다.

3. Julia의 분산-병렬 모델

분산-병렬 컴퓨팅의 구현은 Julia의 Distributed와 Threads 라이브러리를 통해 구현될 수 있다. Julia의 병렬 컴퓨팅의 경우 경량 스레드, 멀티 스레딩, 분산 컴퓨팅, GPU 컴퓨팅을 지원한다. 간단한 예제 코드를 통해 Julia의 분산-병렬 모델 설계에 대해서 설명하고자 한다.

```
1 julia> r = remotecall(rand, 2, 2, 2)
2 julia> s = @spawnat 2 1 .+ fetch(r)
3 julia> fetch(s)
4 2x2 Array{Float64, 2}:
  1.11111 1.50022
  1.16233 1.50482
```

코드 1 Julia Parallel Programming in repl

코드 1을 살펴보면, `remotecall(<Function>, <Process>, <*arg>)` 함수를 통해 명시적인 함수호출과 할당할 프로세스의 개수, 함수의 인자로 특정 프로세스에서 함수를 평가한다. 앞의 설명을 토대로 1번줄의 코드를 해석하게 되면, `rand` 라는 난수를 생성하는 내장 함수를 호출하고 2번째 프로세스에 함수의 인자로 들어온 2와 2를 값으로 받아 2x2행렬에 난수를 생성한다. 2번줄의 코드의 매크로 함수 `@spawnat` 은 비동기적으로 클로저를 실행하는 매크로 함수이다. `fetch` 된 `Future` 객체의 행렬에 1을 더한 값을 `Future` 객체로 변수 s에 할당하게 된다. 마지막으로 `fetch(<FutureObject>)` 함수를 통해 결과값을 반환하며 동작을 마무리한다.

분산-병렬 프로그래밍은 구현하기 어려운 작업으로 알려졌으나, Julia에서의 분산-병렬 프로그래밍은 코드 1을 보면 알 수 있듯이 간단하게 구현될 수 있다, 따라서, Julia를 이용한 분산-병렬 프로그래밍은 비교적 구현하기도 쉬우며, 효율적이라는 것을 알 수 있다.

4. 실험

이 절에서는 Julia 모델과 하이브리드 모델의 성능을 비교하고자 한다. Julia 모델의 경우 Julia 분산 모델의 표준 라이브러리인 Distributed.jl과 Julia 병렬 모델의 표준 라이브러리인 Threads.jl을 사용해 실험에 사용될 모델을 설계했으며, 하이브리드 모델의 경우 앞서 설명했듯이, MPI-OpenMP의 조합으로 설계되었다.

실험에 사용된 CPU 는 Intel 사의 i9-9980HK 모델로서 8코어 16스레드의 CPU 가 사용되었다. 실험 진행의 경우, 프로세스의 수에 따른 확장성을 두고 실험을 진행했으며, 스레드의 경우 프로세스 수 하나당 16스레드를 사용해 실험을 진행했다.

실험에 사용된 데이터는 2차원 데이터 행렬에 10,000,000개의 난수를 생성해 사용되었으며, 군집의 수 k는 5로 설정되었다. K-Means 알고리즘의 최대 반복 수는 따로 고정된 값이 없으며, 기본적으로 설정된 tol의 값인 1e-5만큼 줄어들지 않으면 반복을 멈추고, 프로그램 수행을 종료하는 것으로 설계되었다. 실험 결과를 그래프로 나타내면 그림 2와 같다.

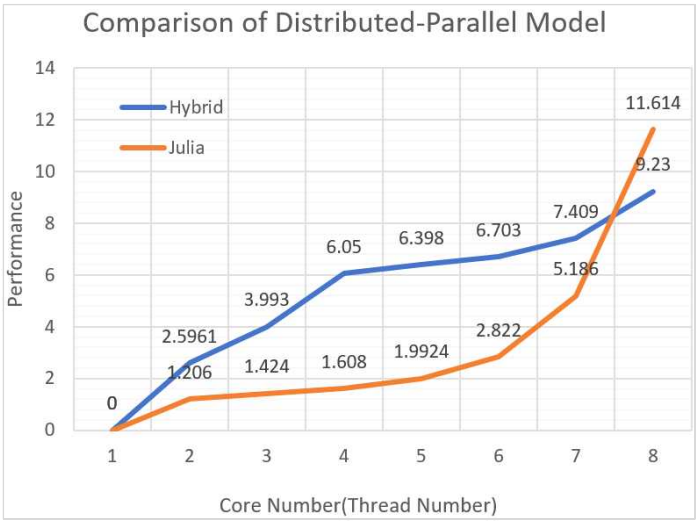

그림 2 모델 간 수행 결과 시각화

그림 2에 따르면, 프로세스가 확장됨에 따라 성능이 점차 개선되어가는 것을 확인할 수 있다. Julia 모델은 프로세스 수가 추가됨에 따라 성능 편차가 점차 커져가는 것을 확인할 수 있다. 프로세스 1개를 사용했을 때와 프로세스 8개를 사용했을 때의 성능 차가 약 12배 정도 차이 났으며, 하이브리드 모델 또한 성능 편차가 점차 커지는 것을 확인할 수 있었지만, 프로세스 1개를 사용했을 때와 8개를 사용했을 때의 성능 차는 약 9배 정도 차이를 보인다.

Julia 모델의 경우, 프로세스 확장에 따른 성능 편차가 큰 편이었으며, 하이브리드 모델은 성능 편차가 크진 않지만, 안정적으로 증가한 것을 확인할 수 있다. 하지만 프로세스 확장에 따른 결과를 볼 때, 상대적으로 Julia 모델이 프로세스 확장에 따라 성능 편차가 높은 것으로 미뤄보아, 알고리즘 수행이 효율적으로 수행되었음을 알 수 있다.

5. 고찰

Julia 모델의 성능 편차가 하이브리드 모델에 비해 높은 이유로는, Julia 모델의 경우 병렬 수행에서 자체 경량 스레드를 사용하기 때문에 문맥 전환(context switching)의 오버헤드가 낮다는 점을 꼽을 수 있다. 반면, 하이브리드 모델의 병렬 수행은 Posix 스레드 기반에서 수행되기 때문에 문맥 전환의 오버헤드가 높은 편이다.

따라서 Julia 모델이 분산-병렬 수행에 있어서 하이브리드 모델보다 효율적이라는 결론을 도출할 수 있다. 다만, 경량 스레드 기반의 분산-병렬모델과 수행 성능 비교를 하지 못한 점과 다른 알고리즘을 대상으로 실험하지 못한 점은 본 연구의 한계점으로 보인다.

6. 결론

전통적인 C 언어 기반의 하이브리드 모델과 Julia 모델과의 K-Means 알고리즘 수행 성능 시간 측정 결과를 통해 성능 비교를 진행해보았다. 비교 결과, 프로세스 확장에 따라 Julia 모델이 하이브리드 모델에 비해 수행 성능의 효율이 높다는 것을 알 수 있었다. 또한, 하이브리드 모델이 Julia 모델보다 코드의 양이 200줄 이상 많은 점과 비교 결과를 토대로 최종적으로 비교해볼 때, 분산-병렬 모델에서 Julia 모델의 분산-병렬 모델이 K-Means 알고리즘을 최적화하기에 적합하게 구현될 수 있다는 결론을 도출할 수 있으며, 분산-병렬 수행에 있어 효율적이라는 점을 알 수 있다.

본 연구의 결과를 토대로, 연구자의 목표는 Julia 모델을 발전시켜 다양한 병렬 알고리즘을 대상으로 한 심도 있는 연구를 진행할 예정이며, 가능성이 풍부한 Julia 언어에도 기여를 하고자 함이 목표이다.

참 고 문 헌

[1] J, Bezanson, A Edelman, S, Karpinski and V. B. Shah, “Julia: A fresh approach to numerical computing,” *Society for Industrial and Applied Mathematics*, Vol. 59, No. 1, pp. 65-98, 2015

[2] H. Ralambondrainy, “A conceptual version of the k-means algorithm,” *Pattern Recognition Letters*, Vol. 16, No. 11, pp. 1147-1157, 1995.

[3] W. Gropp, E. Lusk, N. Doss and A. Skjellum, “A high-performance, portable implementation of the MPI message passing interface standard,” Parallel computing, Vol. 22, No. 6, pp. 789-828, 1996.

[4] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *IEEE Computational Science and Engineering*, Vol. 5, No. 1, pp. 46-55, 1998.

[5] M. J. Chorley and D. W. Walker, “Performance analysis of a hybrid MPI/OpenMP application on multi-core clusters,” *Journal of Computational Science*, Vol. 1, No. 3, pp. 168-174, 2010.

[6] K. Wagstaff, C. Cardie and S. Schroedl "Constrained k-means clustering with background knowledge." *International Conference on Machine Learning*, Vol. 1, pp. 577-584, 2001.