https://doi.org/10.7840/kics.2024.49.2.237

효율적인 스트리밍 데이터 처리를 위한 메타 휴리스틱 알고리즘 기반의 성능 개선에 관한 연구

김 대 광◆, 권 영 우°

A Study on Performance Improvement Based on Meta-Heuristic Algorithms for Efficient Streaming Data Processing

DaeGwang Kim◆, Young-Woo Kwon°

요 약

대량의 스트리밍 데이터를 효율적으로 처리하고 분석해야 할 필요성이 요구되면서 다양한 스트림 처리 플랫폼이 등장했다. 그러나 처리 속도와 사용 효율성 등 여러 성능의 측면을 최적화해야 하는 과제가 여전히 남아있다. 이 연구에서는 고래의 사냥 행동을 시뮬레이션하는 메타휴리스틱 알고리즘인 Whale Optimization Algorithm(WOA)을 사용하여 이와 같은 문제를 해결하기 위한 새로운 접근 방식을 소개한다. 스트림 처리 플랫폼에 적용된 WOA는 기존 알고리즘과 비교해 성능의 개선을 이루었으며 병렬 처리가 중요한 Apache Spark와 같은 분산 컴퓨팅 시스템의 익스큐터 설정을 조정하여 병렬 처리 능력과 전반적인 성능 향상을 보인다. 또한 하이퍼파라미터 최적화 기법인 그리드 탐색을 활용하여 WOA의 하이퍼파라미터를 미세 조정하여 추가적인 성능 향상을 달성한다.

키워드 : 스트리밍 데이터 처리, 메타 휴리스틱 알고리즘, 하이퍼파라미터 최적화
Key Words : Streaming data processing, Metaheuristic Algorithms, Hyperparameter Optimization

ABSTRACT

The need to efficiently process and analyze large amounts of streaming data has led to the emergence of various stream processing platforms. However, the challenge remains to optimize several aspects of performance, such as processing speed and usage efficiency. In this study, we introduce a novel approach to address this problem using the Whale Optimization Algorithm (WOA), a metaheuristic algorithm that simulates the hunting behavior of whales.

Applied to a stream processing platform, WOA achieves performance improvements compared to existing algorithms and shows parallelism and overall performance gains by tuning the executor settings of distributed computing systems such as Apache Spark where parallelism is important. We also leverage grid search, a hyperparameter optimization technique, to fine-tune the hyperparameters of WOA to achieve additional performance gains.

※ 이 논문은 2021년도 과학기술정보통신부의 재원(No. 2021R1A5A1021944)과 교육부의 재원(No. 2021R1I1A3043889)으로 한국연구재단의 지원을 받아 수행된 연구임
◆ First Author : Kyungpook National University, kdk0811@knu.ac.kr, 학생회원
° Corresponding Author : Kyungpook National University, ywkwon@knu.ac.kr, 정회원
논문번호 : 202308-061-B-RN, Received August 24, 2023; Revised October 17, 2023; Accepted November 1, 2023

I. 서 론

스트리밍 데이터 분석의 중요성이 급격히 증가하고 있어, 대용량 실시간 데이터를 신속하게 처리하고 해석하는 방법은 현재 주요 관심사이다. 이에 대응하여 여러 스트림 처리 시스템이 나왔으나, 처리 속도와 자원 효율성을 병행하려는 노력은 복잡한 도전 과제로 남아있다. 높은 처리 속도를 얻기 위해 자원 사용을 늘리는 것과 효율적인 자원 사용을 위해 처리 속도를 제한하는 것 사이에는 상충관계가 있다. 일반적인 스트림 처리 엔진으로 알려진 Apache Storm과 Flink는 마이크로 배치 기반의 Spark Streaming보다 처리 속도가 최대로 15배 이상 빠르다. 그렇지만 Spark Streaming은 노드 장애에 대한 빠른 복구와 데이터 손실의 최소화를 보장한다[1]. 이러한 상황은 높은 처리량과 높은 안정성 간의 균형을 찾아야 함을 의미한다. 이는 각 스트림 처리 시스템이 자체적인 이점과 단점을 가지고 있으며, 이를 일관되게 해결하기 어렵다고 판단된다. 따라서 다양한 목표의 성능 최적화를 동시에 달성하는 것은 쉽지 않음을 나타낸다.

이 문제를 개선하기 위해 본 논문은 저자의 이전 연구인 "Whale Optimization Algorithm을 이용한 효율적인 스트리밍 데이터 처리 향상을 위한 작업 스케줄링 전략 개발에 관한 연구[2]"를 확장하여 다중 목적 최적화를 위한 새로운 방법을 제안한다. 이전 연구에서는 기본적인 WOA 알고리즘의 적용과 그 성능 향상에 중점을 두었으나, 본 논문에서는 추가적으로 하이퍼파라미터 최적화(Hyperparameter Optimization)를 적용하며, FIFO 및 FAIR 스케줄링 알고리즘과의 비교를 통해 성능을 더욱 강화하였다.

메타휴리스틱 알고리즘 중 하나인 Whale Optimization Algorithm(WOA)[3]은 고래의 사냥 행동을 모방하여 수학적으로 모델링된 알고리즘이다. 이 알고리즘은 복잡한 목표들을 동시에 추구하는 최적화 문제에 대해 유용한 해결책을 제시할 수 있다. 여기에서는 성능 평가를 위해 기존의 스케줄링 알고리즘인 FIFO(First-In-First-Out) 및 FAIR 알고리즘과 비교를 통해 성능을 검증하였다. 이를 통해 WOA 기반의 스트림 처리 성능의 효과를 검증한다. 또한 하이퍼파라미터 최적화(Hyperparameter Optimization)를 소개하고 알고리즘에 적용하여 추가적인 성능 개선을 실험을 통해 보인다.

결론적으로, 본 논문에서는 WOA 알고리즘을 스트림 처리 플랫폼에 적용하고 그 효과를 실험 및 비교를 통해 살펴본다. 이를 통해, 작업 처리 속도와 자원 사용 효율성 등의 성능 지표를 동시에 향상시킬 수 있는 새로운 방법을 제시하였다. 이 방법은 기존의 방법들과 비교했을 때 더욱 효과적이라는 것을 보인다. 이러한 발견은 스트림 처리 플랫폼의 성능 개선에 대한 새로운 가능성을 제시할 수 있다.

Ⅱ. 관련연구

최근 몇 년 동안, 다양한 컴퓨팅 환경에서 작업 스케줄링 및 성능 최적화를 위한 메타휴리스틱 알고리즘의 사용은 연구 및 산업 커뮤니티에서 큰 관심을 받고 있다. 자연 현상에서 영감을 얻은 이러한 알고리즘은 복잡한 다중 목표 최적화 문제를 처리하는 데 유망한 결과를 보여주고 있다. 또한 작업 스케줄링과 관련하여 메타휴리스틱 알고리즘을 적용하여 성능을 최적화하는 다양한 연구가 이루어지고 있다.

클라우드 컴퓨팅 환경에서 Genetic Algorithm(GA), Differential Evolution(DE), Simulated Annealing(SA) 등의 알고리즘을 적용해 작업 스케줄링 문제를 해결한 연구[4]가 진행되었다. 그러나 이러한 연구들은 주로 단일 목표 최적화에 중점을 두고 있어, 다중 목표 최적화 문제에 대한 해결책을 제공하지 못하고 있다. 본 연구에서는 WOA를 적용하여 스트림 처리 성능을 높이기 위한 아이디어를 제시하고 실험을 통해 검증하였다.

WOA는 이미 다양한 연구 분야에서 복잡한 최적화 문제나 여러 도메인에서 문제 해결을 위해 사용되고 있는 유용한 알고리즘이다. 예를 들면, 전기 배전 네트워크의 서비스 복원을 위해 WOA를 사용하여 안정성과 효율성을 높이는 연구가 진행되었다[5]. 또한 에너지 효율성을 높이기 위해 흐름 작업 순서에 의존하는 설정에서 하이브리드 WOA를 제시한 연구도 있다[6]. 이 연구에서는 에너지 사용을 최소화하면서 작업 효율을 높이는 방법을 제시한다. 그리고 분산 컴퓨팅 시스템에서 성능 최적화를 위한 연구도 활발히 진행되고 있다. Apache Spark에서 사용자 지정 제약 조건을 적용할 수 있는 경량 리소스 할당 프레임워크 dSpark를 제시한 연구[7]에서 dSpark는 애플리케이션 완료 시간을 익스큐터와 애플리케이션 입력/반복에 대하여 모델링하고 이를 바탕으로 사용자 지정 마감 시간을 고려한 비용 효율적인 리소스 할당 전략을 수립한다.

클라우드 플랫폼에서 Spark의 작업 성능을 최적화하기 위해 Sparker라는 리소스 인식 최적화 전략을 제안한 연구[8]도 있다. Sparker는 클러스터 내의 노드에서 사용 가능한 CPU와 메모리 리소스를 기반으로 익스큐터의 크기를 조정한다. 이를 통해, CPU와 메모리 집약

적인 애플리케이션의 성능을 향상시키고, 실행 시간을 최대 46%까지 줄일 수 있다.

Hadoop 기반의 컴퓨팅 작업 스케줄링 성능 최적화에 관한 연구[9]는 분산 컴퓨팅 시스템의 성능 향상에 중점을 두고 있다. 분산 스트림 처리 시스템(DSPS)에서 작업 스케줄링을 위한 메타휴리스틱 알고리즘의 비교에 대한 최근 연구[10]도 있으며, 이 연구는 다양한 목표를 동시에 달성하는 데 있어 메타휴리스틱 알고리즘의 잠재력을 강조하고 있다. 이 연구는 DSPS의 효율성과 신뢰성 향상을 위한 통찰력을 제공한다.

본 연구는 WOA를 스트림 처리 플랫폼 중 하나인 Apache Spark에 적용하여 성능 향상을 위한 아이디어를 제안한다. 이것은 다중 목표를 동시에 고려하여 작업 속도와 자원 사용 효율을 모두 개선할 수 있다. 또한 실시간 스트리밍 데이터 처리에 특화된 최적화 방법을 제시하여 동적 환경에서의 성능 향상을 이룰 수 있다. 이를 통해 기존 연구의 한계를 극복하고 스트림 처리 효율과 신뢰성을 높일 수 있다.

Ⅲ. 본 론

3.1 메타 휴리스틱 알고리즘(Meta-Heuristic Algorithm)

메타 휴리스틱 알고리즘은 주어진 목적 함수와 관련하여 후보 솔루션을 반복적으로 개선하여 문제를 최적화하는 계산 방법을 뜻한다. 메타 휴리스틱은 최적에 가까운 솔루션을 찾기 위해 탐색 공간을 효율적으로 탐색하도록 하는 전략이다. 메타 휴리스틱은 특정 문제에 국한되지 않으므로 광범위한 최적화 문제에 적용할 수 있다. 메타 휴리스틱 알고리즘은 다양한 자연적, 인공적 프로세스에서 영감을 얻는다. 예로는 유전 알고리즘(Genetic Algorithm), 시뮬레이션 어닐링(Simulated Annealing), 파티클 스웜 최적화(Particle Swarm Optimization), 그리고 고래 최적화 알고리즘(Whale optimization Algorithm) 등이 있다.

메타 휴리스틱 알고리즘의 가장 큰 장점은 기본 문제 영역에 대한 깊은 이해 없이도 복잡한 최적화 문제에 대해 좋은 또는 최적의 솔루션을 찾을 수 있다는 것이다. 크고 복잡한 탐색 공간을 처리할 수 있으며 문제 정의나 요구 사항의 변경에 강점을 보인다.

3.2 Whale Optimization Algorithm(WOA)

고래 최적화 알고리즘(WOA)은 2016년에 Seyedali Mirjalili와 Andrew Lewis가 제안한 생물학적 영감의 메타휴리스틱 최적화 알고리즘[3]이다. 고래의 독특한

그림 1. WOA의 탐색 및 최적화 과정
Fig. 1. Exploring and Optimizing WOA

사냥 전략, 특히 먹잇감을 포위하고 거품망으로 먹이를 주는 기술에서 영감을 얻었다. 고래는 먹이를 찾을 때 여러 마리가 협력하여 먹이를 포위하여 둘러싸고, 먹이에 접근하는 다양한 경로와 속도를 사용한다. 이런 다양한 움직임을 수학적으로 모델링하여 WOA로 제안하고 다양한 목표와 제약 조건 하에서도 효율적인 해를 찾을 수 있다. WOA는 단순성, 손쉬운 구현, 탐색과 착취의 균형으로 인해 복잡한 최적화 문제를 해결하는 데 효과적인 것으로 나타났다.

본 논문에서는 Spark 구성 매개변수 최적화를 위한 WOA의 적용 사례를 소개한다. WOA는 고래의 두 가지 주요 행동, 즉 먹이 포위와 버블넷 공격을 모델링한다. 본 연구에서는 고래 최적화 알고리즘을 스트림 처리 플랫폼의 워커 노드의 익스큐터 설정 조정에 적용하기 위해 알고리즘을 맞춤화한다. 이 알고리즘은 스트림 처리 플랫폼에서 데이터의 크기에 따라 익스큐터의 수와 익스큐터 당 코어 수를 동적으로 조정할 수 있게 설계되었다. 이를 통해 작업 시간과 메모리 사용량을 효과적으로 개선할 수 있다.

고래의 위치는 스트림 처리 설정, 즉 익스큐터 수와 코어 수로 표현된다. 고래의 '적합성'은 목표 함수를 통해 평가되며, 이 목표 함수는 작업 처리 시간과 메모리 사용량을 종합적으로 고려한다. 초기화 단계에서 각 고래(익스큐터 수와 코어 수)는 임의의 위치에 놓인다. 이 위치는 알고리즘의 탐색 공간 내에서 무작위로 선택된다. 각 반복에서 고래는 두 가지 방법으로 위치를 업데이트한다. 먹이 추적과 나선 형태의 움직임으로 표현되는데 이 두 방법은 고래가 적합성에 따라 더 좋은 위치로 이동하도록 돕는다. 각 반복마다 가장 적합한 고래가 선택된다. 이는 다음 반복에서 다른 고래들이 이 최적의 위치를 참고하여 자신의 위치를 업데이트할 때 사용된

다. 알고리즘은 최대 반복 횟수에 도달하거나 설정된 목표 함수 값에 도달할 때까지 계속된다.

알고리즘이 종료되면 가장 적합한 고래의 위치, 즉 최적의 익스큐터 수와 코어 수가 반환된다. 이와 같이 WOA는 복잡한 목표 함수를 가진 문제에 대한 최적의 해를 찾는 데 매우 유용하며, 스트림 처리 플랫폼에서도 뛰어난 성능을 보인다. 이 알고리즘은 스트리밍 데이터의 볼륨, 처리 능력, 그리고 리소스 사용량을 고려한 매개변수 설정이 필요하다.

표 1. 버블넷 사냥의 로그 나선형 방정식
Table 1. The logarithmic spiral equation of a bubble-net attack

X	A new position vector, representing the position of the whale in the next iteration
D	A direction vector or distance vector, representing the distance between an arbitrary solution and the current solution
b	Spiral constant, controlling the shape and density of the spiral
l	A random value between -1 and 1, used to control the angle and direction of the spiral movement
X^*	A random solution vector, representing a random location in the search space
e^{bl}	Control the shape and length of spirals
$cos(2\ \pi l)$	Control the shape and direction of spirals
X	A new position vector, representing the position of the whale in the next iteration

Algorithm 1 Modified Whale Optimization Algorithm for Spark

Input: StreamData, maxIterations, searchSpace
Output: Best config for the number of executors, cores

procedure WOA
  $bestWhale \leftarrow \infty$
  $searchSpace = [(minExecutors, maxExecutors), (minCores, maxCores)]$
  Define $fitness(config, StreamData)$
  Define $updatePosition(whale, bestWhale, a, r, searchSpace)$
  Define $spiralUpdate(whale, bestWhale, b, l, searchSpace)$
  Initialize $population \sim U(searchSpace)$
  for each iteration i in $1 \ldots maxIterations$ **do**
    $dataSize \leftarrow getStreamDataSize(StreamData)$
    Adjust $minExecutors$ and $maxExecutors$ based on $dataSize$
    Adjust $minCores$ and $maxCores$ based on $dataSize$
    $a = 2.0 - i \times \frac{2.0}{maxIterations}$
    for each whale in the population **do**
      Generate $r, p, l \sim U[0, 1]$ and $b = 1$
      if $p < 0.5$ **then**
        $x' = updatePosition(whale, bestWhale, a, r, searchSpace)$
      else
        $x' = spiralUpdate(whale, bestWhale, b, l, searchSpace)$
      end if
      $f = fitness(whale.position, StreamData)$
    end for
    $currentBestWhale = \min(population)$
    $bestWhale = \min(bestWhale, currentBestWhale)$
  end for
  return the integer values of $bestWhale.position$
end procedure

그림 2. 고래 최적화 알고리즘[2]
Fig. 2. Whale Optimization Algorithm

$$\overrightarrow{X}(t+1) = \overrightarrow{X^*} - \overrightarrow{A} \cdot \overrightarrow{D} \quad (1)$$

(1)은 고래가 다음 반복에서 리더를 따라가기 위해 현재 위치를 업데이트하는 방법을 나타낸다. 다음의 식 (2)는 버블넷 공격을 수학적으로 모델링하여 방정식으로 나타낸 것이다.

$$\overrightarrow{X'} = \overrightarrow{D} \cdot e^{bl} \cdot cos(2\ \pi l) + \overrightarrow{X} \quad (2)$$

이 알고리즘은 문제를 다중 목표 최적화 문제로 취급하고 WOA를 사용하여 최적의 솔루션을 찾음으로써 Spark 구성 매개변수를 조정하는 효율적인 방법을 제공한다.

3.3 WOA를 적용한 스트림 처리 성능 분석

WOA 적용의 성능 비교를 위해 일반적으로 Spark에서 사용되는 두 가지 스케줄링 알고리즘인 FIFO 및 FAIR 알고리즘과 또다른 메타 휴리스틱 알고리즘인 ACO에 대해 WOA의 성능을 비교 평가했다. 측정 메트릭에는 작업 실행 시간의 평균값 및 중앙값과 평균 메모리 사용량이 포함되었다. 실험은 다양한 크기의 웹 로그 데이터셋에 대한 집계, 카운트, 윈도우 작업을 수행하는 Spark 스트리밍 애플리케이션을 각각 30회 반복하여 실행하고 결과를 측정하고 FIFO, FAIR 알고리즘 및 또다른 메타휴리스틱 알고리즘과의 비교를 위해 Ant Colony Optimization(ACO) 알고리즘과의 성능 비교로 수행되었다.

그 결과 아래 그림 3에 보이는 것처럼 WOA를 사용했을 때 성능의 개선이 이루어진 것으로 나타났다. 구체적으로, WOA는 평균 작업 시간이 FIFO에 비해 약 14.01%, FAIR에 비해 약 24.64%, ACO와 비교했을 때는 각각 13.64%와 24.31%가 단축되었다.

메모리 사용량 측면에서는 그림 4에서와 같이 WOA의 경우, FIFO에 비해 약 9.03%, FAIR에 비해서는 20.98% 감소했으며, ACO와 비교시 8.14%와 20.19%

그림 3. 알고리즘들 간의 작업 시간 비교
Fig. 3. Comparison of the job times between algorithms

표 2. 알고리즘들 간의 평균 작업 시간 값 비교
Table 2. Comparison of the job times between algorithms

Algorithms	Average job time (sec)
FIFO	141.84
FAIR	161.83
ACO	122.47 (13.64%, 24.31%)
WOA	121.96 (14.01%, 24.64%)

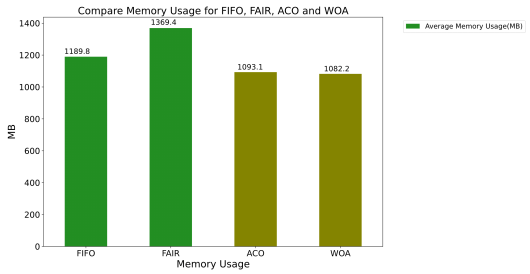

그림 4. 알고리즘들 간의 메모리 사용량 비교
Fig. 4. Comparison of the memory usage between algorithms

표 3. 알고리즘들 간의 평균 메모리 사용량 비교
Table 3. Comparison of the memory usage between algorithms

Algorithms	Average memory usage (MB)
FIFO	1189.8
FAIR	1369.4
ACO	1093.1 (8.14%, 20.19%)
WOA	1082.2 (9.03%, 20.98%)

감소한 것으로 나타났다.

3.4 병렬 처리와 메타 휴리스틱 알고리즘 적용 사이의 효율성

병렬 처리는 Spark와 같은 분산 컴퓨팅 시스템의 성능에 중요한 역할을 한다. Spark에서 병렬 처리 정도는 주로 익스큐터 수와 익스큐터당 코어 수라는 두 가지 매개변수에 의해 영향을 받는다. 최적의 시스템 성능을 달성하려면 이러한 매개변수의 균형을 맞추는 것이 필수적이다. 익스큐터 수와 코어 수는 Spark 클러스터의 병렬 처리 능력을 결정짓는 중요한 요소이다. 익스큐터 수가 너무 많으면 통신 오버헤드가 과도하게 발생할 수 있으며, 익스큐터당 코어 수가 너무 많으면 리소스 경합이 발생할 수 있다. 두 시나리오 모두 각 작업의 성능에 악영향을 미치고 결과적으로 시스템 처리량을 저하시킬 수 있다.

이 문제를 해결하기 위해 WOA가 사용된다. WOA는 복잡한 최적화 문제를 해결하는 데 효과적인 메타휴리스틱 알고리즘이다. Spark의 병렬 처리 문제에 WOA를 적용하면, 익스큐터 수와 코어 수 사이의 균형을 찾을 수 있다. WOA는 여러 해의 탐색을 통해 리소스 활용과 통신 오버헤드가 조화를 이루는 최적화된 구성을 찾아낸다. 이 과정에서, WOA는 각 해의 적합도를 평가하고, 최적의 해를 찾아가는 과정을 반복한다. WOA의 적용을 통해, Spark의 병렬 처리 성능이 향상될 수 있다. 익스큐터 수와 코어 수 사이의 최적 균형을 찾으면, 통신 오버헤드를 최소화하고 리소스 경합을 방지할 수 있다. 이로 인해, 각 작업의 처리 속도가 향상되고, 전체 시스템의 처리량이 증가한다.

병렬 처리와 WOA 적용의 효율성은 Spark와 같은 분산 컴퓨팅 시스템의 성능 최적화에 중요한 역할을 한다. WOA를 통해 병렬 처리의 복잡한 문제를 해결하면, 시스템의 전반적인 성능과 효율성을 향상시킬 수 있다. 이러한 접근 방식은 Spark뿐만 아니라 다른 분산 컴퓨팅 시스템에도 적용될 수 있으며, 병렬 처리의 최적화에 대한 새로운 연구 방향을 제시한다.

이러한 접근 방식의 효율성을 입증하는 것은 WOA를 활용하는 Spark 애플리케이션이 달성한 처리량으로 그림 4를 보면, WOA는 FIFO와 FAIR 및 ACO 알고리즘을 모두 능가함을 보여준다.

같은 실험에 대하여 FIFO와 비교했을 때 WOA는 처리량이 21.56% 증가했으며, FAIR와 비교하였을 때

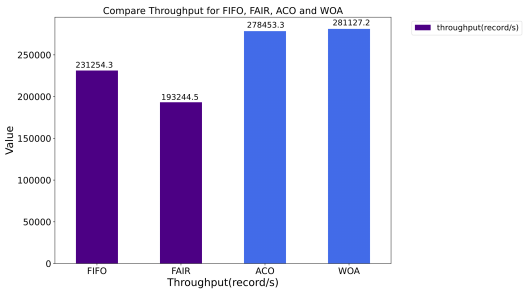

그림 5. 알고리즘들 간의 평균 처리량 비교
Fig. 5. Comparison of the throughput between algorithms

표 4. 알고리즘들 간의 평균 처리량 비교
Table 4. Comparison of the throughput between algorithms

Algorithms	Average memory usage (MB)
FIFO	231254.3
FAIR	193244.5
ACO	278453.3 (20.41%, 44.09%)
WOA	281127.2 (21.56%, 45.47%)

45.47% 증가하였다. ACO와 비교하였을 때는 각각 20.41% 및 44.09% 더 좋은 처리율을 보여준다.

이 비교 분석은 병렬 처리가 처리량에 미치는 영향을 명확하게 보여준다. 또한, Spark에서 WOA와 같은 최적화된 스케줄링 접근 방식을 채택할 때 얻을 수 있는 이점을 보여준다. 그 결과 리소스의 균형을 더욱 세밀하게 조정하여 처리량을 개선함으로써 기존의 FIFO 및 FAIR 알고리즘을 능가하는 성능을 발휘한다.

3.5 하이퍼파라미터 최적화(HPO)

최적화 알고리즘에서 하이퍼파라미터는 모델의 구조 또는 학습 프로세스 수행 방식을 정의하는 매개변수이다. 이러한 하이퍼파라미터를 조정하는 것은 최상의 성능을 발휘하는 모델 또는 최적화 프로세스를 찾는 데 매우 중요하다. 이러한 하이퍼파라미터를 조정하는 일반적인 기법은 그리드 탐색으로, 각 하이퍼파라미터에 대해 가능한 값의 집합을 정의하고 이러한 값의 모든 조합을 체계적으로 시도하여 최상의 값을 찾는다. 하이퍼파라미터는 알고리즘 성능을 결정짓는 중요한 요소이다.

WOA는 상대적으로 하이퍼파라미터의 수가 적어 그리드 탐색과 같은 HPO를 적용하기 수월하다. 이는 WOA의 계산 복잡성을 낮추고 빠른 수렴 속도를 가능하게 한다. 반면에, ACO의 하이퍼파라미터는 복잡하고 다양한 옵션을 제공한다. 이로 인해 다양한 솔루션을 탐색할 수 있는 능력이 있다. WOA와 ACO 사이에는 명확한 트레이드오프가 있다. WOA는 계산 복잡성이 낮지만, 솔루션의 다양성이 떨어질 수 있다. 반면 ACO는 계산 복잡성이 높지만 더 다양한 솔루션을 탐색할 수 있다[11].

이러한 특성은 특히 여러 목적을 동시에 최적화해야 하는 문제에서 중요하다. 결론적으로, WOA와 ACO는 각자의 장단점을 가지며, 이는 하이퍼파라미터 튜닝의 복잡성과 효율성에 영향을 미친다. 따라서 특정 문제에 어떤 알고리즘이 더 적합한지를 판단할 때 이러한 특성을 고려해야 한다. 본 연구에서는 그리드 탐색 기법을 사용하여 WOA와 ACO의 하이퍼파라미터를 최적화한다. 최적화되는 WOA의 하이퍼파라미터는 population size와 max iterations 및 b(스파이럴 업데이트에 필요한 상수)이다.

ACO의 경우 numberofAnts(개미의 수), evaporationRate(페로몬 증발 속도), alpha(페로몬 중요도), beta(경로 길이 중요도), initialPheromone(초기 페로몬 레벨), maxIterations(최대 반복 횟수)이다.

그리드 탐색 프로세스는 수학적으로 다음과 같이 정의할 수 있다.

표 5. WOA와 ACO의 하이퍼파라미터 비교
Table 5. Comparison of the Hyperparameter between WOA and ACO

WOA	ACO
populationSize	numberofAnts
maxIterations	maxIterations
b	evaporationRate
	alpha

$\Theta = \{\theta_1, \theta_2\}$를 하이퍼파라미터 집합으로, 여기서 θ_1은 모집단 크기이고 θ_2 는 최대 반복 횟수이며, $G_i = \{g_i1, g_i2, \ldots, g_ini\}$를 θ_i에 대해 가능한 값의 집합이라고 한다. 그리드 탐색의 목표는 목적 함수 $f(\Theta)$를 최소화하는 하이퍼파라미터 조합 $\Theta^* = \{\Theta^*_1, \Theta^*_2\}$를 찾는 것이다.

$$\Theta^* = arg\ min\ \Theta\ f(\Theta)$$

WOA의 성능은 다른 최적화 알고리즘과 마찬가지로 하이퍼파라미터의 선택에 따라 크게 달라진다. 여기에는 고래 개체군의 크기와 알고리즘이 실행할 최대 반복 횟수, 스파이럴 업데이트에 사용되는 파라미터가 포함된다. 이러한 매개변수를 부적절하게 선택하면 최적이 아닌 솔루션이나 불필요한 계산 비용이 발생할 수 있다. 하이퍼파라미터 최적화를 위해 사용할 수 있는 몇 가지 기법이 있으며, 각 기법에는 장단점이 있다.

본 연구에서는 여러 가지 이유로 그리드 탐색 방법을 사용하기로 결정했다. 첫째, 그리드 탐색은 전체 하이퍼파라미터 공간을 체계적으로 탐색하여 잠재적인 파라미터 조합을 놓치지 않는다. 이러한 철저한 탐색은 복잡한 스케줄링 문제에 있어서 최적의 해를 찾을 가능성을 높인다. 둘째, 본 연구의 주요 목적은 스트림 처리 성능을 개선하는 것이다. 이 프로세스의 성능은 하이퍼파라미터 설정에 따라 복잡하고 비선형적인 방식으로 변할 수 있다. 예를 들어, 일부 하이퍼파라미터 설정은 작업 처리 속도를 높일 수 있지만 메모리 사용량을 증가시킬 수 있다. 이런 상호작용을 고려할 때, 그리드 탐색은 복잡한 성능 지표 간의 균형을 잘 맞출 수 있는 방법론이라고 판단했다.

3.5.1 그리드 탐색을 적용한 최적화

WOA를 최적화하기 위해 그리드 탐색을 적용하였다. 목표로 삼은 하이퍼파라미터는 표1에서 찾을 수 있는 populationSize와 maxIterations 및 b로 총 3개이며,

각각 최적화 알고리즘이 고려해야 하는 솔루션의 수와 알고리즘이 수행해야 하는 최대 반복 횟수를 나타낸다. 비교를 위해 ACO 알고리즘에서 표 5의 6개 매개변수에 대해 최적화를 진행했다.

그리드 탐색은 다음과 같이 시행된다. gridSearchParameters 배열에 지정된 매개변수의 각 조합에 대해 WOA를 실행한다. runOptimizations 함수는 특정 실행에서 찾은 최적의 솔루션의 적합성을 계산하여 반환한다.

여기에서는 적합도 값이 낮을수록 최적의 솔루션에 더 가까운 근사치를 나타낸다. 현재 실행의 적합도가 가장 잘 기록된 적합도보다 더 나은 것으로 판명되면 현재 실행의 매개변수를 새로운 최적 매개변수로 저장하고 최적 적합도를 현재 실행의 적합도로 업데이트한다.

이 프로세스는 가능한 모든 매개변수 조합이 탐색될 때까지 반복된다. 그림6와 그림7은 그리드 탐색을 적용하지 않았을 때와 적용했을 때의 WOA 및 비교를 위해 ACO 알고리즘을 적용한 스트림 처리 애플리케이션의 작업 시간과 메모리 사용량을 측정하고 비교한 그림이다. 결과적으로 그리드 탐색을 적용했을 때 WOA의 작업 시간의 경우 32.13%, 개선되었고, 메모리 사용량은 평균 21.58%의 개선을 확인하였다. ACO의 경우 각각 작업 시간 평균값은 31.39 개선됨을 보였고, 메모리 사용량의 경우 21.82% 개선을 보였다. 이 결과는 그리드 탐색을 적용했을 때 알고리즘의 하이퍼파라미터를 최적화하여 성능을 향상시킬 수 있음을 보여준다.

그리드 탐색은 알고리즘의 하아퍼파라미터 공간을 체계적으로 탐색한다. 이러한 탐색은 WOA와 ACO 같은 복잡한 문제에 대해 최적의 하이퍼파라미터 조합을 찾아내는 데 특히 유용하다. 따라서 WOA는 ACO에 비해 적은 수의 매개변수를 최적화하는 것으로도 큰 폭의 성능 개선을 달성할 수 있음을 보인다.

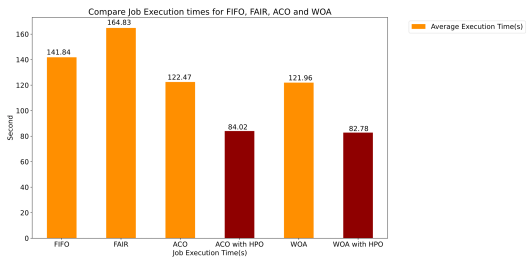

그림 6. HPO 적용 전과 후 알고리즘들 간의 작업 시간 비교
Fig. 6. Comparison of job times between algorithms before and after applying HPO

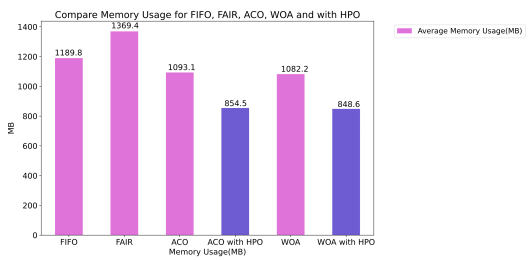

그림 7. HPO 적용 전과 후 ACO의 메모리 사용량 비교
Fig. 7. Comparison of the memory usage between algorithms before and after applying HPO

표 6. HPO 적용 전과 후 알고리즘들 간의 작업 시간 및 메모리 사용량 비교
Table 6. Comparison of job times between algorithms before and after applying HPO

Metrics (Algorithm)	Before	After	Improvement
Average Job Time(WOA)	122.47 sec	**84.02 sec**	**31.39%**
Memory Usage(WOA)	1093.1 MB	**854.5 MB**	**21.82%**
Average Job Time(ACO)	121.96 sec	**82.78 sec**	**32.13%**
Memory Usage(ACO)	1082.2 MB	**848.6 MB**	**21.58%**

Ⅳ. 결 론

본 연구에서는 스트리밍 데이터 처리의 성능 개선을 위한 도구로서 Whale Optimization Algorithm(WOA)을 적용하는 연구를 진행하고 스트림 처리 플랫폼 환경에서 WOA를 적용하여 성능을 비교하고 분석하였다. 비교 분석을 통해 WOA를 적용했을 때 FIFO 및 FAIR와 같은 기존 스케줄링 알고리즘에 비해 작업 시간, 메모리 사용량 및 처리량과 같은 메트릭에서 개선된 성능을 제공한다는 것을 보였다. 이로써 WOA의 구현 및 적용이 메타 휴리스틱 알고리즘의 스트림 처리의 성능 개선에 좋은 해결책이 될 수 있음을 보였다. 또한 하이퍼파라미터 최적화 기법인 그리드 탐색을 적용하여 WOA의 하이퍼파라미터를 조정하는 것이 알고리즘의 성능에 영향을 미칠 수 있음을 보였다. 향후 연구에서는 스트림 처리와 병렬 처리 성능 개선을 위해 다양한 성능 지표들을 측정하고 최적화를 위한 연구를 진행할 예정이다. 이를 위해 다양한 데이터 분포와 워크로드 조건에서 WOA의 성능을 평가하고 최적화한다.

결론적으로, 이 연구는 스트림 처리에서 WOA의 적용 가능성과 장점을 보임으로써 연구의 발전 가능성을 보였다. 이번 연구는 보다 효율적인 스트림 처리 시스템을 설계하고 개선하는 데 도움이 될 수 있으며, 이를

통해 효율적인 자원 활용과 성능 향상을 기대할 수 있으며, 스트림 처리의 병렬 처리 성능 개선에 대한 이해와 해결책을 제공할 것으로 기대된다.

References

[1] M. A. Lopez, A. G. P. Lobato, and O. C. M. B. Duarte, "A performance comparison of open-source stream processing platforms," *2016 IEEE GLOBECOM*, pp. 1-6, 2016. (https://doi.org/10.1109/GLOCOM.2016.7841533)

[2] D. G. Kim and Y.-W. Kwon, "A study on developing a task scheduling strategy for efficient streaming data processing using the Whale Optimization Algorithm," in *Proc. KICS Summer Conf.*, pp. 1760-1761, 2023.

[3] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in Eng. Softw.*, pp. 51-67, 2016. (https://doi.org/10.1016/j.advengsoft.2016.01.008)

[4] M. N. Aktan and H. Bulut, "Metaheuristic task scheduling algorithms for cloud computing environments," *Concurrency and Computation: Practice and Experience*, pp. e6513, 2022. (https://doi.org/10.1002/cpe.6513)

[5] B. Qiu and Y.-S. Oh, "A study on service restoration scheme for distribution networks using whale optimization algorithm," *The Trans. KIEE*, pp. 30-36, 2023. (http://doi.org/10.5370/KIEE.2023.72.1.30)

[6] D. M. Utama, "The hybrid whale optimization algorithm: A new metaheuristic algorithm for energy-efficient on flow shop with dependent sequence setup," *J. Phys.: Conf. Ser.*, vol. 1569, no. 2, p. 022094, 2019. (http://10.1088/1742-6596/1569/2/022094

[7] M. T. Islam, S. Karunasekera, and R. Buyya, "dSpark: Deadline-based resource allocation for big data applications in apache spark," *2017 IEEE 13th Int. Conf. e-Science*, pp. 88-98, 2017. (https://doi.org/10.1109/eScience.2017.21)

[8] N. Garg and D. Janakiram, "Sparker: Optimizing spark for heterogeneous clusters," *2018 IEEE Int. Conf. CloudCom*, pp. 1-8, 2018. (https://doi.org/10.1109/CloudCom2018.2018.00017)

[9] Y. Li and X. Hei, "Performance optimization of computing task scheduling based on the Hadoop big data platform," *Neural Comput. & Applic.*, p. 12, 2022. (https://doi.org/10.1007/s00521-022-08114-3) (http://dx.doi.org/10.4018/IJDST.2020100102)

[10] D. Kim, A. Wu, and Y.-W. Kwon, "Comparison of meta-heuristic algorithms for task scheduling in distributed stream processing," *2022 IEEE 27th Pacific Rim Int. Symp. Dependable Computing (PRDC)*, pp. 252-255, 2022. (https://doi.org/10.1109/PRDC55274.2022.00041)

[11] Y. A. Ali, E. M. Awwad, M. Al-Razgan, and A. Maarouf, "Hyperparameter search for machine learning algorithms for optimizing the computational complexity," *Processes*, vol. 11, no. 2, p. 349, 2023. (https://doi.org/10.3390/pr11020349)

김 대 광 (DaeGwang Kim)

2020년 2월 : 부경대학교 IT융합응용공학과 졸업

2022년 3월~현재 : 경북대학교 컴퓨터학부 석사과정

<관심분야> 분산시스템, 빅데이터, 스트림 처리, IoT

권 영 우 (Young-Woo Kwon)

2003년 2월 : 경북대학교 컴퓨터과학과 졸업
2005년 2월 : 광주과학기술원 정보통신공학과 석사
2014년 7월 : Virginia Tech, 컴퓨터과학과 박사
2014년 8월~2017년 6월 : Utah State University, 교수
2017년 8월~현재 : 경북대학교, 컴퓨터학부 교수
<관심분야> 분산시스템, 빅데이터, 인공지능, IoT, 지진조기경보, 재난 ICT
[ORCID:0000-0003-0625-8232]