일반논문 (Regular Paper)
방송공학회논문지 제24권 제6호, 2019년 11월 (JBE Vol. 24, No. 6, November 2019)
https://doi.org/10.5909/JBE.2019.24.6.1122
ISSN 2287-9137 (Online) ISSN 1226-7953 (Print)

하둡 기반 대규모 작업처리 프레임워크에서의 Adaptive Parallel Computability 기술 연구

김 직 수[a)‡]

A Study on Adaptive Parallel Computability in Many-Task Computing on Hadoop Framework

Jik-Soo Kim[a)‡]

요 약

본 연구팀에서는 YARN 기반의 하둡 플랫폼에서 대규모의 태스크들로 구성된 Many-Task Computing(MTC) 응용들을 효율적으로 지원할 수 있는 신규 프레임워크로서 MOHA(Mtc On HAdoop)를 연구/개발해왔다. MTC 응용들은 수십만 개에서 수백만 개 이상의 대규모 태스크들로 구성되고 각 응용별로 자원의 사용 패턴이 다를 수 있기 때문에, 전체적인 시스템 성능 향상을 위해 MOHA-TaskExecutor (MTC 응용 태스크를 실행하는 주체)의 Adaptive Parallel Computability 기술 연구를 수행하였다. 이는 한 번에 하나의 태스크를 실행하던 기존의 처리 모델을 고도화하여 하나의 TaskExecutor가 동시에 여러 개의 태스크들을 실행함으로써 YARN Container의 병렬 컴퓨팅 능력을 극대화하기 위함이다. 이를 위해 각각의 TaskExecutor들이 "독립적이고, 동적으로" 동시에 실행시키는 MTC 응용 태스크들을 조정할 수 있도록 하였으며, 최적의 동시 실행 태스크 숫자를 찾기 위해서 Hill-Climbing 알고리즘을 활용하였다.

Abstract

We have designed and implemented a new data processing framework called MOHA(Mtc On HAdoop) which can effectively support Many-Task Computing(MTC) applications in a YARN-based Hadoop platform. MTC applications can be composed of a very large number of computational tasks ranging from hundreds of thousands to millions of tasks, and each MTC application may have different resource usage patterns. Therefore, we have implemented MOHA-TaskExecutor(a pilot-job that executes real MTC application tasks)'s Adaptive Parallel Computability which can adaptively execute multiple tasks simultaneously, in order to improve the parallel computability of a YARN container and the overall system throughput. We have implemented multi-threaded version of TaskExecutor which can "independently and dynamically" adjust the number of concurrently running tasks, and in order to find the optimal number of concurrent tasks, we have employed Hill-Climbing algorithm.

Keyword : Many-Task Computing, YARN, Hadoop, MOHA, Parallel Computability

a) 명지대학교 공과대학 컴퓨터공학과(Department of Computer Engineering, Myongji University)
‡ Corresponding Author : 김직수(Jik-Soo Kim)
E-mail: jiksoo@mju.ac.kr
Tel: +82-31-330-6436
ORCID: https://orcid.org/0000-0002-0104-4617

※ This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2019R1A2C1005360), and the Ministry of Education (No. 2017S1A3A2066319).
· Manuscript received August 29, 2019; Revised November 21, 2019; Accepted November 21, 2019.

I. 서 론

1. 배경 및 필요성

기존의 클러스터 컴퓨팅 자원 관리 및 작업 스케줄링 역할을 수행해온 HTCondor[1], PBS[2], LoadLeveler[3], SGE[4] 등과 같은 로컬 배치스케줄러(Local Batch Scheduler)들은 분산 클러스터 또는 슈퍼컴퓨팅 환경에서 계산 집적형(Compute-intensive)이며 상대적으로 실행시간이 긴 독립적인 작업들로 구성된(loosely-coupled) HTC (High-Throughput Computing) 응용, 또는 MPI (Message Passing Interface)[5]와 같은 통신 인터페이스를 활용하는 밀착 결합형(tightly-coupled) 병렬 프로그램을 실행하는 형태의 응용들로 구성된 HPC (High-Performance Computing) 분야를 성공적으로 지원해왔다. 그러나 최근 다양한 과학 응용 분야들(천문학, 약학, 물리학 등)을 중심으로 기존의 HTC 또는 HPC 분야의 기술들을 통해 지원하기 어려운 문제들이 나타나고 있음을 제시하고, 이러한 HTC와 HPC의 중간 개념의 새로운 컴퓨팅 패러다임으로서 MTC (Many-Task Computing)[6]가 미국 Argonne National Lab을 중심으로 주창되어 왔다. 이들 그룹은 세계적인 슈퍼컴퓨팅 학회인 SC에서 정기적으로 워크숍[7]을 개최하면서 MTC에 관련된 기술 교류를 선도해나가고 있다.

MTC 응용들은 주로 대규모의 작업 계산(수백만에서 수십억 개의 태스크), 상대적으로 짧은 작업 실행 시간(초/분 단위), 데이터 집적성(태스크 당 수십 MB 정도의 I/O), 상대적으로 큰 작업 수행 시간 편차, 그리고 파일을 통한 통신 방식 등의 특징을 가지고 있다. 이는 대규모의 계산 작업들을 상대적으로 짧은 시간 안에 고성능으로 처리해야한다는 점에서 (# of operations/month)가 주요 성능 측정치인 HTC와 차별화되며, MPI에 기반한 통신 방식을 활용하는 HPC 응용들과도 차이점을 가지게 된다. 이러한 MTC 응용들을 효과적으로 지원하기 위해서는 기존 배치 스케줄러를 뛰어넘는 고성능 작업 배포, 동적인 자원 할당 및 로드밸런싱, 안정성 및 다양한 자원들의 효율적인 통합 등이 필수적인 기능이라 할 수 있다. MTC 응용은 각각의 태스크들이 요구하는 I/O 처리량은 상대적으로 많지 않지만(64MB 데이터 블록에 기반한 기존의 하둡 응용 대비), 동시에 대량의 태스크들을 고성능으로 처리해야하고, 이들이 파일을 통해서 통신을 한다는 점에서, 또 다른 패턴의 데이터 집약형 워크로드라고 할 수 있다.

이러한 MTC 응용들의 효율적인 지원을 위해 미국 Argonne National Lab에서는 기존의 로컬 배치 스케줄러의 성능을 뛰어넘을 수 있는 대규모 작업 처리 프레임워크인 Falkon(Fast and Light-weight tasK executiON framework)[8,9]을 개발하였다. Falkon은 기존 배치 스케줄러에서의 긴 큐 대기시간을 줄이기 위한 Multi-level Scheduling 기술과, 대규모 태스크 배치(Dispatch) 성능을 위한 streamlined task dispatching, 그리고 데이터 집적도가 높은 MTC 응용 지원을 위한 data caching 및 data-aware scheduler 기능들을 결합하여 초당 15,000개 이상의 태스크 배치 성능을 보이고 있다.

하둡(Hadoop)[10]은 빅데이터 처리 플랫폼의 대명사로서 아파치 프로젝트를 통해 오픈소스로 개발되고 있다. 하둡은 대용량 데이터를 분산 저장할 수 있는 파일시스템(HDFS: Hadoop Distributed File System)과 병렬 처리 프로그래밍 모델(MapReduce)을 제공하고 있다. 데이터는 미리 정해진 크기(기본 64MB)의 블록 단위로 HDFS 노드들에 분산 저장되며 고가용성을 위해서 중복저장(기본 3개)하여 데이터 손실 가능성을 극복하고 안정성을 높이고 있다. 이렇게 분산 저장된 데이터 파일들의 위치 정보를 활용(Data Locality)하여 대규모 데이터 처리를 위한 MapReduce가 프로그래밍 모델 및 런타임 시스템으로서 활용되고 있다.

기존의 하둡 1.0은 이러한 HDFS와 MapReduce 기반의 배치 프로세싱에 최적화된 빅데이터 처리 플랫폼이었으나, YARN[11,12]의 등장과 함께 기존 MapReduce 형태의 배치 작업뿐만 아니라 인터액티브, 스트리밍, 그래프 처리 등과 같이 다양한 데이터 처리 프레임워크들이 하둡 플랫폼이 제공하는 자원을 공유할 수 있는 형태로 발전하고 있다. 이에 따라 최근에는 기존의 고성능컴퓨팅 분야 (HPC: High-Performance Computing)에서 주로 활용되던 MPI (Message Passing Interface) 응용들도 YARN 기반의 하둡 플랫폼에서 지원하기 위한 연구들이 활발하게 진행되고 있다[13,14,15].

본 연구팀은 이러한 하둡 2.0에서의 새로운 데이터 처리 프레임워크로서 국가 거대과학분야(수백만 개에서 수천만 개 이상 태스크들로 구성)에서의 MTC 응용(예: 신약개발,

신약재창출 등)을 하둡을 통해서 효율적으로 지원할 수 있는 MOHA(Mtc On HAdoop) 프레임워크[16]를 연구/개발하고 있다. 이를 통해 멀티 응용 플랫폼으로 진화하고 있는 하둡 생태계에 신규 프레임워크로서 대규모 계산과학 응용을 지원할 수 있는 MOHA를 추가하고, 하둡 생태계 및 빅데이터 플랫폼의 확장에 기여할 수 있을 것이다.

2. 주요 연구 내용

실제 MTC 과학 응용의 적용을 준비하는 과정에서 주로 대규모의 태스크들로 구성된 신약개발 분야의 응용들의 분석을 수행하였다. 신약 개발 분야는 새로운 신약 후보군을 대규모의 도킹 시뮬레이션을 통해서 찾는 “Virtual Screening” (e.g. 1개의 단백질 파일(disease)에 대한 3,133,839개의 리간드 파일(drug) 도킹 실험) 분야[17]와, 이미 알려진 약물의 또 다른 파생 효과를 찾아내는 Drug Repositioning (신약재창출: e.g. 1개의 약물(ligand)에 대해서 총 12,119종류의 단백질 구조(disease)로부터 얻어진 36,000여 개의 단백질 결합부위를 도킹 실험) 분야[18]로 구분될 수 있다. 이 중에서 상대적으로 도킹 시뮬레이션 실행 시간이 짧은 신약재창출 분야를 주요 MTC 응용으로 선정하였다. 신약재창출 시뮬레이션을 수행하기 위해서는 대규모의 도킹 연산(AutoDock Vina[19] 활용)이 필요하게 되며(예를 들어 기존에 알려진 95개의 약물에 대해 12,000여개의 단백질과의 교차 도킹 실험의 경우에도 110만 번 이상의 시뮬레이션 필요), 각 작업이 수십 초 정도 걸린다는 점에서 비교적 짧은 수행시간을 가지는 많은 양의 작업들을 고성능으로 처리해야하는 Many-Task Computing 성격을 지니고 있다.

본 논문에서는 이러한 신약재창출 응용을 활용하여, MTC 응용 실행 최적화를 위한 MOHA-TaskExecutor(MTC 응용 태스크 실행 주체) Adaptive Parallel Computability 기술 연구를 수행하였다. 이는 대규모의 태스크들로 구성된 MTC 응용 실행의 최적화를 위해 각각의 TaskExecutor들의 “Parallel Computability”(동시에 실행시키는 태스크의 수)를 동적으로 조정할 수 있는 기술을 의미한다. 즉, 한 번에 하나의 AutoDock-Vina 도킹(docking) 태스크를 실행하던 기존의 처리 모델을 고도화하여 하나의 TaskExecutor가 동시에 여러 개의 도킹 태스크들을 실행함으로써 하둡 YARN Container의 병렬 컴퓨팅 능력을 극대화하기 위함이다.

그런데, 각각의 MOHA-TaskExecutor들이 동시에 실행시킬 수 있는 MTC 응용 태스크들의 숫자는 실행되는 응용의 자원 활용 패턴(예: CPU 집약형, 메모리 집약형, I/O 집약형 등), TaskExecutor가 구동되는 물리 노드의 HW/SW 사양, 하나의 물리 노드들에서 동시에 실행되는 다른 TaskExecutor들에 의한 간섭 현상 등 다양한 변수들에 의해서 영향을 받을 수 있다. 또한, 동시에 실행시키는 태스크들의 숫자를 늘려감에 따라 어느 시점이 되면 최적의 성능을 나타내지만, 점점 더 숫자를 늘려갈수록 오히려 성능 저하가 발생할 수 있을 것이다. 따라서 각각의 MOHA-TaskExecutor들이 “독립적이고, 동적으로” 동시에 실행시키는 MTC 응용들의 태스크들을 조정할 수 있도록 하였으며, 최적의 동시 실행 태스크 숫자를 찾기 위해서 Hill-Climbing 알고리즘을 활용하였다.

II. 관련 기술 및 연구

1. 신약재창출 응용

최근 다양한 계산과학 분야(신약개발, 고에너지 물리, 핵물리 등)에서 대규모의 계산과 데이터 접근에 대한 요구가 증대하고 있으며, 이러한 서로 독립적인 많은 양의 작업들을 고성능으로 처리하기 위한 효율적인 분산 컴퓨팅 기술에 대한 중요성이 높아지고 있다. 신약재창출(Drug Repositioning) 시뮬레이션 분야도 이러한 분산 컴퓨팅을 필요로 하는 응용 분야 중 하나로서, 이미 안전성과 선행데이터가 갖춰진 기존 약물의 새로운 적응증을 규명하는 것을 의미한다. 이는 특정 질병의 치료를 목표로 하여 타깃 단백질(protein)에 영향을 줄 수 있는 다량의 화합물(ligand, potential new drugs)들의 도킹(docking) 연산을 수행하는 기존의 신약개발 분야(Virtual Screening)와는 달리 다량의 단백질 구조 정보를 필요로 한다는 점에서 정반대의 접근 방법이라고 할 수 있다.

이러한 신약재창출 시뮬레이션을 수행하기 위해서는 대규모의 도킹 연산이 필요하게 되며(예를 들어 기존에 알려

진 95개의 약물에 대해 12,000여개의 단백질과의 교차 도킹 실험의 경우에도 110만 번 이상의 시뮬레이션 필요), 각 작업이 수십 초 정도 걸린다는 점에서 비교적 짧은 수행시간을 가지는 많은 양의 작업들을 고성능으로 처리해야하는 Many-Task Computing 성격을 지니고 있다. 본 연구팀에서는 이러한 상대적으로 도킹 시뮬레이션 실행 시간이 짧은 신약재창출 분야를 시범 적용 대상으로 선정하였다. 이는 AutoDock Vina 프로그램을 활용하여 이미 알려진 약물의 또 다른 파생 효과를 찾아내기 위해 1개의 약물(ligand)에 대해서 총 12,119 종류의 단백질 구조(disease)로부터 얻어진 36,000여 개의 단백질 결합부위를 도킹 시뮬레이션하는 다수의 태스크들로 구성된다.

2. MOHA 프레임워크 구조

MOHA는 대부분의 하둡 YARN 어플리케이션들과 같이 YARN Client로 구현되는 MOHA Client, Application Master로 구현되는 MOHA Manager, 그리고 MTC 태스크들의 실행을 담당하는 MOHA TaskExecutor로 구성되어 있다(그림 1 참조).

MOHA Client는 사용자로부터 MTC 응용에 대한 작업 제출 의뢰를 받아 다수의 태스크들을 생성해서 MOHA 작업 큐에 넣어두고, 필요한 데이터와 실행 파일 등을 HDFS에 업로드하는 역할을 수행한다. MTC 응용은 수십만 개에서 수백만 개에 이르는 대규모 태스크들로 구성될 수 있으므로 이들을 안정적이고 확장성 있게 저장할 수 있는 분산 메시지 큐(Distributed Message Queue)로서 Apache Kafka[20,21]와 ActiveMQ[22]를 활용하고 있다.

그림 1에서 보는 것처럼, Kafka 클러스터도 YARN 어플리케이션의 형태로 하둡 클러스터에서 실행되고 있으며, MOHA Client가 사용자 작업을 제출할 때에 필요한 MOHA 작업 큐를 Kafka 클러스터(또는 ActiveMQ) 상에서 생성하고, 작업 기술 언어에 명시되어 있는 응용의 입력 파일, 실행 파일 등의 구조에 따라 필요한 태스크들을 생성해서 MOHA 작업 큐에 집어넣게 된다. 이렇게 생성된 MOHA 작업 큐와 관련 태스크들은 Kafka 클러스터(또는 Active-

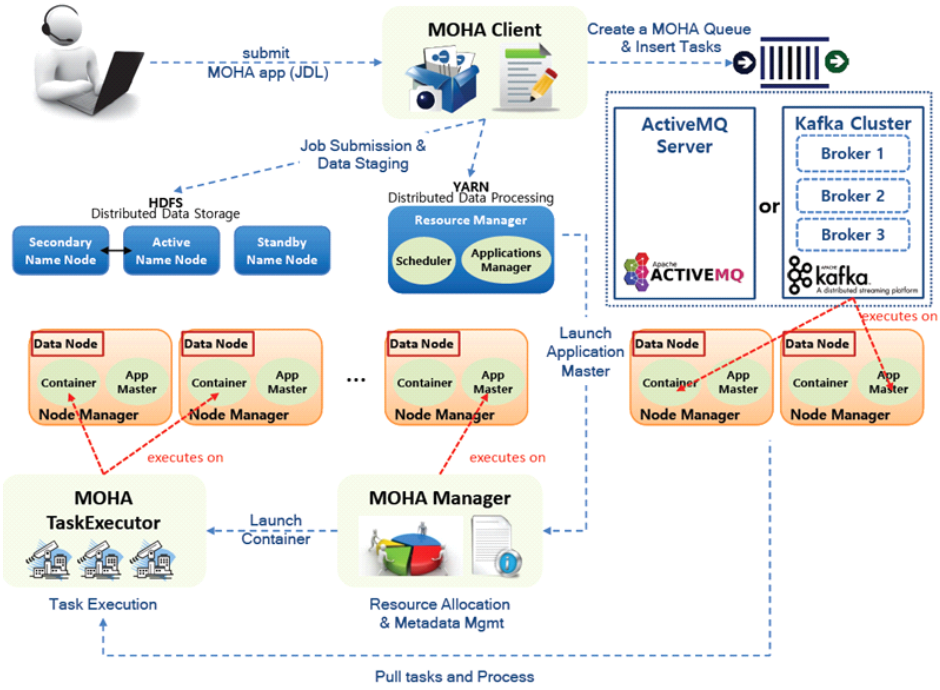

그림 1. MOHA 프레임워크 시스템 구조도
Fig. 1. MOHA Framework System Architecture

MQ)에서 자체적으로 관리하게 되며, 생성된 MOHA 작업 큐 정보를 MOHA Manager와 TaskExecutor에게 알려줌으로써 각각의 태스크들이 MOHA TaskExecutor가 실행되는 Container 상에서 성공적으로 수행되게 된다.

Kafka는 메시지들을 일정한 크기의 파티션(Partition)들로 분할하여 다중 노드들에 저장함으로써 고성능의 메시지 배포 기능을 지원하고 있다. 그런데, Kafka의 경우 정적인 메시지 파티셔닝으로 인한 실제 MTC 응용 태스크들을 수행 시에 전체적인 로드밸런싱에 문제가 발생함을 발견하였다. 이에 본 연구팀에서는 Kafka보다는 중앙 집중적인 아키텍처를 통해 전통적인 큐를 구현하고 있는 Apache ActiveMQ를 MOHA에 같이 적용하고, 이를 통해 신약재창출 응용 실행 시간의 단축과 전체적인 태스크들의 처리 로드를 효과적으로 분산시킬 수 있음을 확인하였다. 이에 따라서 좀 더 다양한 형태의 MTC 응용들을 지원할 수 있도록 작업 제출 인터페이스를 개선하고 고도화하였다. 또한, Apache Kafka의 장점도 살려서 MTC 커뮤니티에서 중요하게 고려하는 고성능 태스크 배포 성능 측정 실험도 수행함으로써 MOHA를 단일한 형태의 작업 제출 인터페이스를 갖는 두 개의 프레임워크(MOHA-Kafka, MOHA-ActiveMQ)로 이원화하였다.

MOHA Client는 MTC 응용 실행에 필요한 파일들(입력 파일, 실행 파일 등)을 HDFS에 업로드하게된다. 이렇게 업로드된 HDFS URI 정보를 태스크에 명시함으로써 MOHA TaskExecutor가 어떤 노드에서 실행되더라도 MTC 응용 실행에 필요한 파일들을 HDFS로부터 다운로드 받을 수 있게 하였다. 향후 MOHA TaskExecutor 자체를 Container에 할당할 때에 실행에 필요한 파일들의 Data Locality를 고려해서 MOHA Manager가 YARN Resource Manager에게 Container 할당을 요청할 수 있게 함으로써 전체적인 수행 시간을 향상시키는 연구도 필요할 것이다. MOHA Manager에 의해 실행된 각각의 MOHA TaskExecutor들은 Kafka 클러스터에 생성되어 있는 MOHA 작업 큐로부터 태스크들을 가져와서 이를 해석하고 실행에 필요한 파일들을 다운로드함으로써 MTC 응용 실행 환경을 구성한다. MTC 응용의 실행이 끝나고 나면 결과 파일들을 압축해서 HDFS에 업로드 한 이후 TaskExecutor는 종료하게 됨으로써 전체적인 MTC 응용 실행 워크플로우를 구현하였다.

이러한 형태의 작업 처리 방식은 상대적으로 실행 시간이 짧은 MTC 응용들을 YARN에서 일일이 컨테이너를 할당 받아 실행하는 오버헤드를 극복하고자 하는데 주목적이 있다[16]. 대규모 태스크들에 대한 자원 할당 시간을 단축하

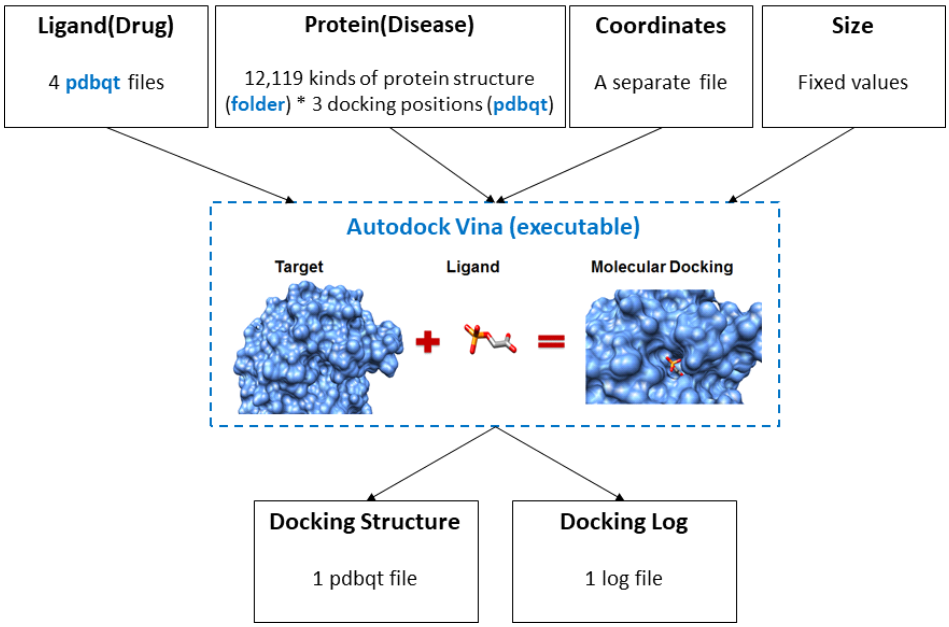

그림 2 신약재창출 응용의 실행 구조
Fig. 2. Execution Structure of Drug Repositioning Application

고 고성능으로 태스크들을 실행하고자 별도의 분산 메시지 큐를 관리하게 된다.

III. MOHA-TaskExecutor Adaptive Parallel Computability 기술 연구

본 연구팀에서는 국가 거대과학 분야의 MTC 응용인 신약재창출을 활용하여 MTC 응용 실행 최적화를 위한 MOHA-TaskExecutor Adaptive Parallel Computability 기술 연구를 수행하였다. 아래 신약재창출 응용의 실행 구조도(그림 2 참조)에서 볼 수 있듯이, 실제 MTC 과학 응용들은 다양한 형태의 입력 파라미터(Parameter)들을 가지게 된다.

예를 들어, Ligand, Protein 입력 파일들은 특정 디렉토리에 저장되어 있어서 이들 간의 조합을 통해 도킹 실험을 수행하는 태스크들을 생성하는 데 활용된다(Parameter Sweep 기능). 즉, 1개의 Ligand 파일에 36,000개의 Protein 파일들을 매핑하면 총 36,000개의 도킹 시뮬레이션 태스크가 생성되어야 한다. 반면, Coordinates 파일과 Size 값들은 고정된 형태의 파라미터들로 볼 수 있다. 이에 따라 MOHA에서는 신약재창출과 같이 입력 데이터 파일들의 조합에 주로 기반하여 실행 태스크들을 생성할 수 있는 형태의 MTC 과학 응용들과, Microbenchmark(“sleep 0” 태스크들로 구성)와 같이 단순한 shell script 형태의 응용들을 지원할 수 있도록 작업 제출 인터페이스를 개선하였다.

이러한 신약재창출 응용을 활용하여 대규모의 태스크들로 구성된 MTC 응용 실행의 최적화를 위해 각각의 MOHA-TaskExecutor들의 “Parallel Computability”를 동적으로 조정할 수 있는 기술에 대한 연구/개발을 수행하였다. 이는 한 번에 하나의 AutoDock-Vina 도킹(docking) 태스크를 실행하던 기존의 처리 모델을 고도화하여 하나의 TaskExecutor가 동시에 여러 개의 MTC 응용 태스크들을 실행할 수 있도록 함으로써 YARN Container의 병렬 컴퓨팅 능력을 극대화하기 위함이다.

그런데, 이러한 MOHA-TaskExecutor의 Parallel Computability를 적절한 선에서 조정하는 것은 쉽지 않은 문제일 수 있다. 이는 하나의 YARN Container에서 할당된 한정된 자원(CPU/Memory) 상에서 병렬로 실행될 수 있는 MTC 응용 태스크들의 숫자가 너무 적으면 “Under Resource Utilization”이, 반대로 너무 많으면 “Over Resource Utilization”이 발생할 수 있기 때문이다. 또한, 이러한 자원의 효율적인 활용도 역시 각각의 TaskExecutor가 실행되는 환경(실행 노드의 H/W 사양, 동시에 실행되는 다른 응용 태스크들의 특성, 현재 가용한 자원량 등)에 따라 달라질 수 있을 것이다. 따라서 각각의 TaskExecutor들이 주어진 자원 가용량(CPU/Memory 등)에 최적화된 병렬 태스크들

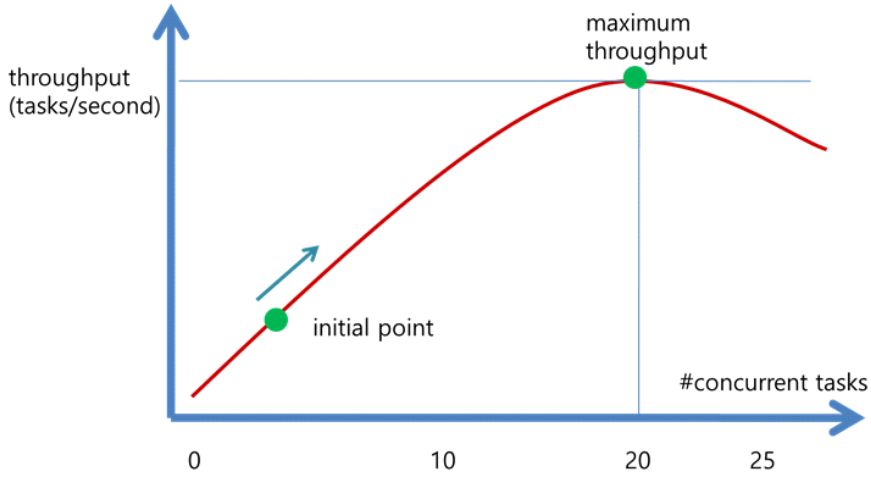

그림 3. Adaptive Parallel Computability를 위한 Hill-Climbing Algorithm
Fig. 3. Hill-Climbing Algorithm for Adaptive Parallel Computability

의 숫자를 적절하게 결정해야 전체적인 시스템 활용률과 Throughput을 높일 수 있게 된다.

본 연구팀은 이에 따라 각각의 MOHA-TaskExecutor들이 "독립적이고, 동적으로" 동시에 실행시키는 MTC 응용들의 태스크들을 조정할 수 있도록 구현하였으며, 이러한 최적의 동시 실행 태스크 숫자를 찾기 위해서 그림 3과 같은 Hill-Climbing 알고리즘을 활용하였다.

하둡 YARN을 통해서 할당되는 Container들은 보통 CPU와 메모리 단위로 자원 할당이 일어나게 되고, 특히 메모리 자원에서 대해서는 YARN의 Node Manager가 엄격하게 관리하는 편이다. 반면, CPU 자원에 대해서는 일정 수준 이상의 Over-provisioning을 허용하고 있다. 심지어 하둡 커뮤니티에서는 YARN이 CPU Isolation 기능을 기본적으로는 제공하고 있지 않다고도 한다. 만약 좀 더 높은 수준의 CPU Isolation을 위해서는 Docker와 같은 Container 기술(YARN 기본 Container와 다름)을 활용하거나 별도의 세팅[1]을 해야 한다고 알려져 있다. 이는 빅데이터 응용의 특성상 CPU보다는 메모리와 디스크에 대한 연산이 상대적으로 더 많이 일어나기 때문으로 사료된다. 따라서 Hill-Climbing 알고리즘을 적용 시에 무작정 병렬 실행 태스크들의 숫자를 늘리지 말고, 현재 TaskExecutor가 실행되는 Container에 할당된 메모리 용량에 기반하여 조정할 필요가 있게 된다. 이는 실제로 YARN Container에 할당된 메모리 용량을 지속적으로 초과하여 사용하는 것이 모니터링되면 YARN Node Manager가 해당 Container를 강제 종료시키기 때문이다.

MOHA-TaskExecutor의 Adaptive Parallel Computability 기술 구현을 위해 그림 4와 같이 "Multi-threaded" 버전의 MOHA-TaskExecutor를 개발하였고, 각각의 Thread는 하나의 MTC 응용 태스크와 매핑되게 함으로써 태스크 관리의 편이성을 구현하였다.

각각의 MOHA-TaskExecutor에서 총 가용한 Thread들의 Pool을 생성해두고, 현재 설정된 동시 실행 태스크들의 숫자에 맞추어서 일부 Thread들은 sleep 상태로 들어가게 구

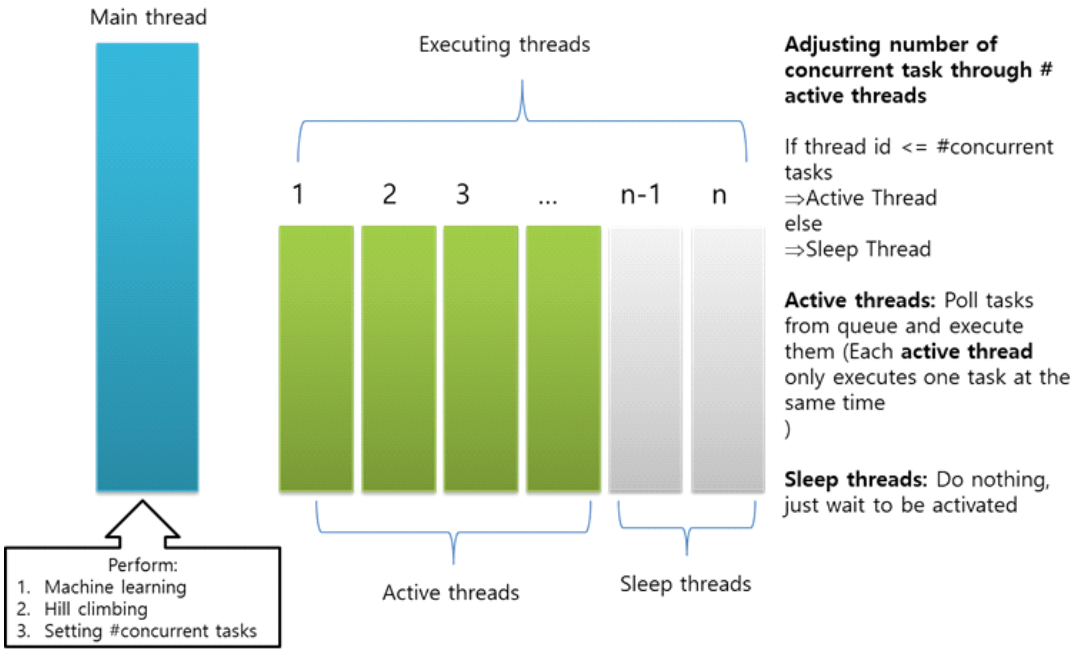

그림 4. TaskExecutor Adaptive Parallel Computability 기술 구현
Fig. 4. Implementation of TaskExecutor Adaptive Parallel Computability

1) https://hortonworks.com/blog/apache-hadoop-yarn-in-hdp-2-2-isolation-of-cpu-resources-in-your-hadoop-yarn-clusters/

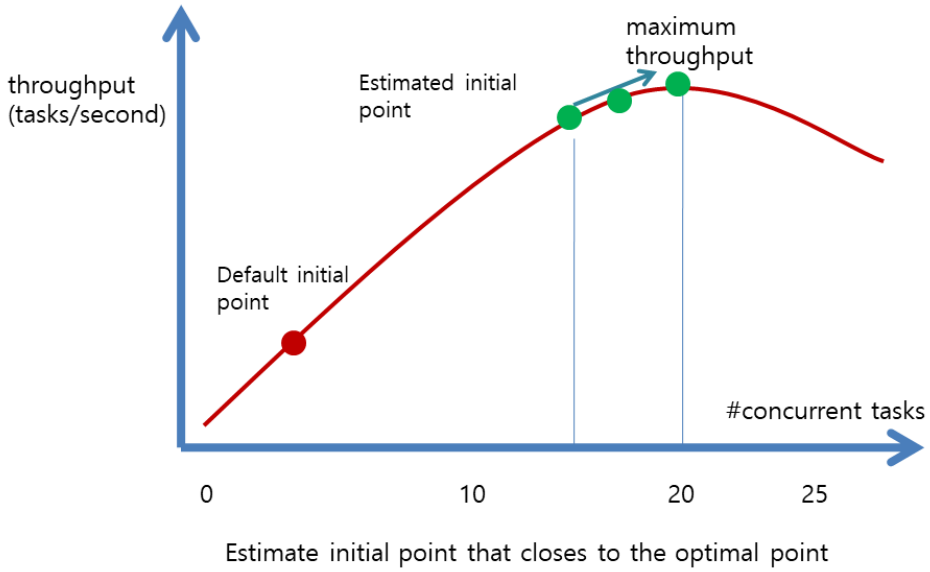

그림 5. Hill-Climbing 알고리즘에서의 시작 포인트 예측을 통한 오버헤드 감축
Fig. 5. Reducing the overhead by estimating the initial point of Hill-Climbing Algorithm

현하였다.

본 연구팀에서 구현한 Hill-Climbing 알고리즘의 특성 상 최적의 동시 실행 태스크들의 숫자를 찾는 데 있어서 오래 걸릴 수 있다는 단점이 존재한다. 이는 한 번에 몇 계단씩 오르는지를 구현하느냐에 따라서 때로는 너무 느리게, 때로는 너무 빨라서 Ping-Pong 현상이 발생할 수 있기 때문이다. 만약 어떠한 MTC 응용이 실행 될 때에 되도록 최적의 태스크 병렬 숫자에 가깝도록 초기 값을 설정할 수 있다면 이와 같은 성능상의 오버헤드를 극복할 수 있을 것이다(그림 5 참조). 이를 위해 본 연구팀에서는 기존에 실행되었던 MTC 응용들에 대한 다양한 데이터들(# of CPUs, Memory amount, I/O 등)을 수집하여 이를 프로파일링함으로써 초기 수행 속도를 향상시킬 수 있는 방법을 연구하고 있다.

IV. 실험 결과

본 연구팀에서는 MOHA-TaskExecutor Adaptive Parallel Computability 기술의 타당성을 1차적으로 검증하기 위해 서로 다른 H/W 사양으로 구성된 하둡 클러스터(표 1

표 1. MOHA-Testbed 구성 내역
Table 1. MOHA-Testbed Configurations

Hostname	Hadoop Components	Hardware Spec.
hdp01.kisti.re.kr	Master (ResourceManager, NameNode), Slave(NodeManager, DataNode)	Dell PowerEdge R630 Rack Servers 2 * Intel Xeon E5-2620v3 (2.4GHz, 6-Core) 64GB (4 x 16GB RDIMM, 2133MT/s)
hdp02.kisti.re.kr	Slaves (NodeManager, DataNode)	
hdp03.kisti.re.kr		
hdp04.kisti.re.kr		Dell PowerEdge R630 Rack Servers 2 * Intel Xeon E5-2609v3 (1.9GHz, 6-Core) 64GB(4 x 16GB RDIMM, 2400MT/s)
hdp05.kisti.re.kr		
hdp06.kisti.re.kr		

MOHA-Testbed 참조)에서 일련의 실험들을 수행하였다. 실험 테스트베드로서 KISTI[2]에 설치되어있는 총 6대의 노드들로 구성된 MOHA-Testbed를 활용하였다. MOHA-Testbed는 hdp01~hdp03에 이르는 3개의 노드들과, hdp04~06에 이르는 3대의 노드들의 집합들이 서로 다른 H/W 구성(특히 CPU 사양)을 가지고 있다. 이 때문에 두 개의 집합에 속하는 노드들이 서로 다른 태스크 처리량을 보일 수 있다. 하둡을 구성하는 YARN과 HDFS의 Master 프로세스들은 모두 hdp01 노드에서 실행되고, hdp01~06에 이르는 6개의 노드들을 Slave들로 구성하였다.

이에 기반하여 서로 하드웨어 사양이 다른 MOHA-Testbed의 hdp01, hdp04 노드들에서 아래와 같이 최적의 동시 실행 태스크들을 Hill-Climbing 알고리즘을 활용하여 찾는 실험을 수행하였다.

그림 6에서 볼 수 있듯이 동적으로 동시 실행하는 MTC 응용 태스크들의 숫자를 조정할 수 있는 기능을 MOHA- TaskExecutor에 구현함으로써 주어진 환경에 기반하여 최대 성능을 낼 수 있는 태스크들의 수를 찾아갈 수 있음을 확인하였다. hdp01 노드의 경우 hdp04 노드보다 하드웨어 사양이 높아서 좀 더 많은 수의 신약재창출 도킹 태스크들을 실행할 수 있음을 보여주고 있다.

또한, 이러한 각 YARN Container에서의 Parallel Computability에 대한 기본 데이터를 수집하기 위해서 AutoDock-Vina 응용 태스크들을 활용하여 그림 7과 같은 실험을 수행해보았다.

그림 7에서는 MOHA-Testbed에서 가용한 총 6개의 노드들(hdp01, hdp02, hdp03, hdp04, hdp05, hdp06)에서 각각 1개의 MOHA-TaskExecutor(YARN Container)들만 실행시키고, 병렬로 실행하는 AutoDock 도킹 태스크들의 숫자를 늘려가면서 성능을 측정한 결과를 보여주고 있다. 6개의 노드들에서 모두 일정 숫자 이상의 병렬 도킹 태스크들이 실행되게 되면 성능이 한계점에 도달하는 것을 알 수

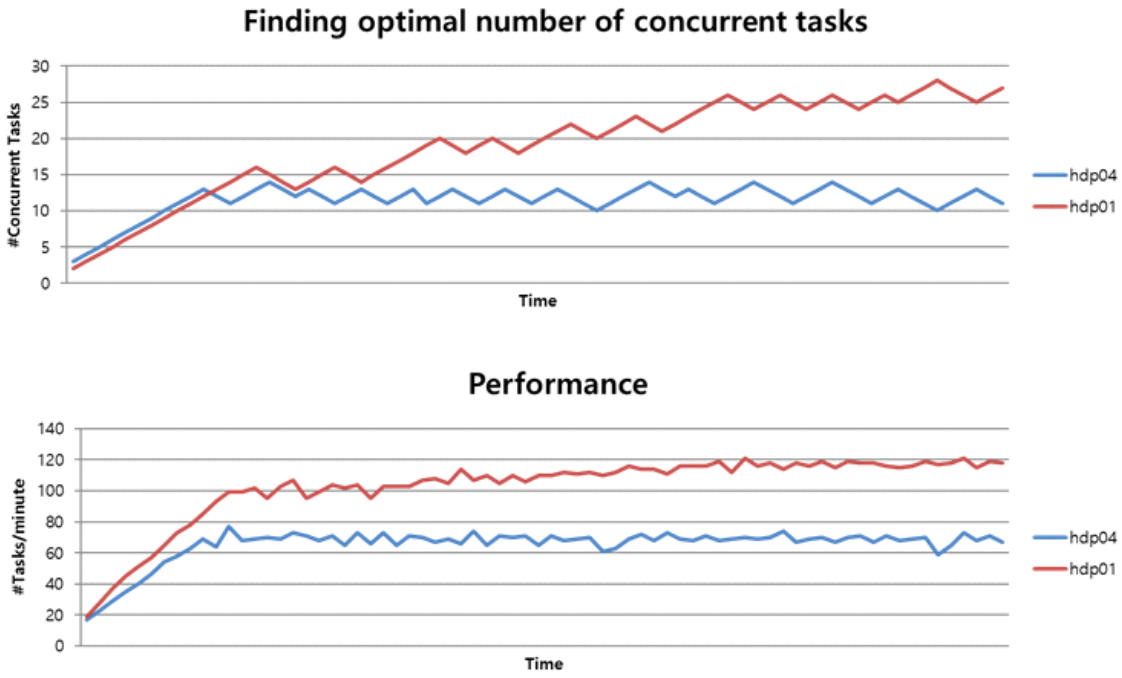

그림 6. MOHA-TaskExecutor Adaptive Parallel Computability 실험
Fig. 6. Experimental Results of MOHA-TaskExecutor's Adaptive Parallel Computability

2) 한국과학기술정보연구원: https://kisti.re.kr/

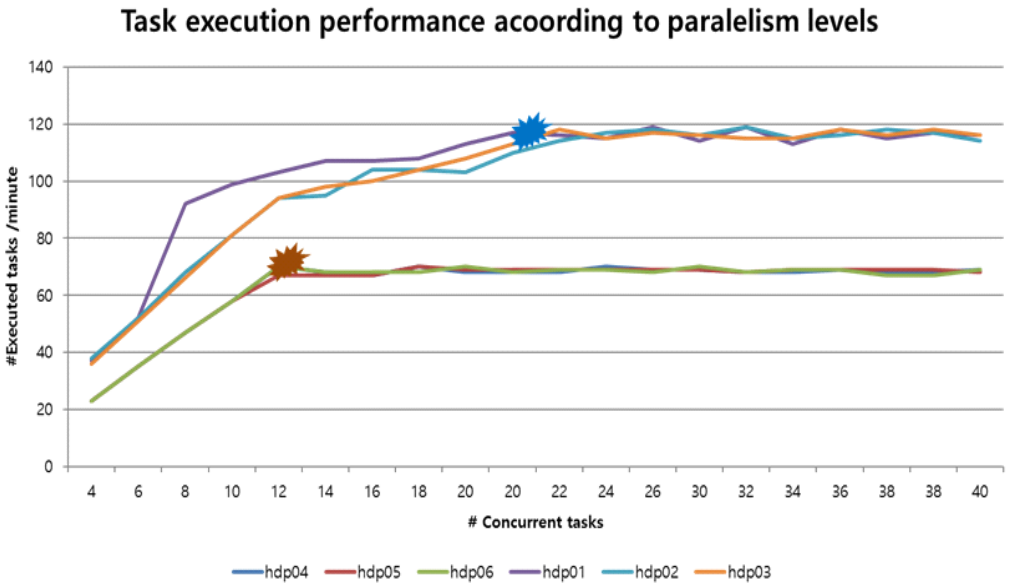

그림 7. Parallel Computability 측정 초기 실험 결과
Fig. 7. Initial Experimental Results of measuring Parallel Computability

있다. 또한, hdp01, hdp02, hdp03 노드들이 나머지 노드들인 hdp04, hdp05, hdp06 노드들보다 전체적인 처리 성능이 더 높게 나오는 것이 관측되었다. 이는 hdp01, hdp02, hdp03 노드들의 하드웨어 사양(특히 CPU)이 더 높기 때문에 나타나는 현상이다.

따라서 각각의 MOHA-TaskExecutor들이 동시에 실행시킬 수 있는 MTC 응용 태스크들의 숫자는 실행되는 응용의 자원 활용 패턴(예: CPU 집약형, 메모리 집약형, I/O 집약형 등), TaskExecutor가 구동되는 물리 노드의 HW/SW 사양, 하나의 물리 노드들에서 동시에 실행되는 다른 Task-Executor들에 의한 간섭 현상 등 다양한 변수들에 의해서 영향을 받을 수 있다. 또한, 동시에 실행시키는 태스크들의 숫자를 늘려감에 따라 어느 시점이 되면 최적의 성능을 나타내지만, 점점 더 숫자를 늘려갈수록 오히려 성능 저하가 발생할 수 있을 것이다. 이러한 동시에 실행되는 태스크들의 숫자는 사용자가 대략적인 값을 사전에 입력할 수도, 또는 응용의 시작 시점에 임의로 정하기도 매우 어렵기 때문에 결국 각각의 TaskExecutor들이 “독립적이고, 동적으로” 동시에 실행시키는 MTC 응용들의 태스크들을 조정할 수 있도록 해야 한다.

V. 결 론

본 논문에서는 AutoDock-Vina에 기반한 신약재창출 응용을 대상으로 MTC 응용 실행 최적화를 위해 MOHA-TaskExecutor Adaptive Parallel Computability 기술 연구 내용을 제시하였다. 이는 한 번에 하나의 태스크를 실행하던 기존의 처리 모델을 고도화하여 하나의 TaskExecutor가 동시에 여러 개의 태스크들을 실행함으로써 YARN Container의 병렬 컴퓨팅 능력을 극대화하기 위함이다. 이를 위해 각각의 TaskExecutor들이 “독립적이고, 동적으로” 동시에 실행시키는 MTC 응용들의 태스크들을 조정할 수 있도록 하였으며, 최적의 동시 실행 태스크 숫자를 찾기 위해서 Hill-Climbing 알고리즘을 활용하였다. AutoDock-Vina 응용 태스크들을 활용한 실험 결과, 각 실행 노드의 H/W 및 S/W 환경에 따라 각각의 TaskExecutor들이 독립적으로 최적의 동시 실행 태스크들의 숫자를 찾아갈 수 있음을 확인하였다. 향후 다양한 응용들에 기반한 실험들을 수행함으로써 MOHA의 신규 기능인 Adaptive Parallel Computability 기술의 우수성을 보이고자 한다.

YARN의 등장과 함께 기존의 MapReduce 기반 Batch

Processing 중심의 하둡 1.0 플랫폼은 다양한 형태의 데이터 처리 워크플로우(Batch, Interactive, Online, Streaming 등)들을 지원할 수 있는 “Multi-Use Data Platform”으로 진화하고 있다. YARN은 클러스터 자원 관리와 어플리케이션 라이프 사이클 관리라는 두 가지 핵심 기능을 분리하고 새로운 추상화 레이어를 구성함으로써 다양한 데이터 처리 어플리케이션들의 수용이 가능하게 하였다. 본 연구팀에서도 Many-Task Computing 응용을 하둡을 통해서 효율적으로 지원할 수 있는 MOHA 프레임워크를 연구/개발함으로써, 하둡 생태계 및 빅데이터 플랫폼의 확장에 기여할 수 있을 것이다.

참 고 문 헌 (References)

[1] D. Thain, T. Tannenbaum, and M. Livny, “Distributed computing in practice: the Condor experience”, Concurrency and Computation: Practice and Experience, Volume 17, Issue 2-4, pp. 323‐356, February. 2005.

[2] B. Bode, D. M. Halstead, R. Kendall, Z. Lei, and D. Jackson, “The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters”, Proceedings of the Usenix, Proceedings of the 4th Annual Linux Showcase & Conference, Nov. 2000.

[3] IBM Tivoli Workload Scheduler LoadLeveler, Available at http://www03.ibm.com/systems/software/loadleveler/

[4] W. Gentzsch, “Sun Grid Engine: Towards Creating a Compute Power Grid”, Proceedings of the 1st IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001), May 2001.

[5] J. J. Dongarra, S. W. Otto, M. Snir, and D. Walker, “A message passing standard for MPP and workstations”, Communications of the ACM, Volume 39, Issue 7, pp. 84-90, July 1996.

[6] I. Raicu, I. Foster and Y. Zhao, “Many-Task Computing for Grids and Supercomputers”, Proceedings of the IEEE/ACM Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS'08), 2008.

[7] Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers, http://datasys.cs.iit.edu/events/MTAGS16/

[8] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, “Falkon: a Fast and Light-weight tasK executiON framework,” Proceedings of the 2007 ACM/IEEE conference on Supercomputing (SC’07), Nov. 2007.

[9] Ioan Raicu et al., “Middleware Support for Many-Task Computing”, Cluster Computing, Volume 13 Issue 3, September 2010.

[10] Apache Hadoop: https://hadoop.apache.org/

[11] Vinod Kumar Vavilapalli et. al., “Apache Hadoop YARN: yet another resource negotiator”, Proceedings of the 4th annual Symposium on Cloud Computing (SOCC'13), October 2013.

[12] Arun C. Murthy et. al., “Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2”, Addison-Wesley, 2014.

[13] Lu, X., Liang, F., Wang, B., Zha, L., Xu, Z., “DataMPI: extending MPI to Hadoop-like big data computing”, Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium (IPDPS ’14), May 2014.

[14] Xu, L., Li, M., Butt, A.R., “GERBIL: MPI+YARN”, Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), May 2015.

[15] Ye, J., Chow, J.H., Chen, J., Zheng, Z., “Stochastic gradient boosted distributed decision trees”, Proceedings of the 18th ACM conference on Information and knowledge management (CIKM’09), Nov. 2009.

[16] Kim, J.S., Nguyen, C., Hwang, S., “MOHA: many-task computing meets the big data platform”, Proceedings of the IEEE 12th International Conference on eScience (eScience 2016), Oct. 2016.

[17] J. Lee, J-W. Choi, J. Shin, K-T. No, “Trends in Computer Aided Drug Discovery, Next Generation”, COMMUNICATIONS OF THE KOREA INFORMATION SCIENCE SOCIETY, Vol. 31, No. 8, pp. 35-54, 2013.

[18] Ashburn, T.T., Thor, K.B., “ Drug repositioning: identifying and developing new uses for existing drugs”, Nature Reviews Drug Discovery, Volume 3, Issue 8, pp. 673-683, 2004.

[19] AutoDock Vina: molecular docking and virtual screening program: Available at http://vina.scripps.edu/

[20] J. Kreps, N. Narkhede, and J. Rao. “Kafka: A distributed messaging system for log processing”, Proceedings of NetDB’11, June 2011.

[21] Apache Kafka: https://kafka.apache.org/

[22] Apache ActiveMQ: http://activemq.apache.org/

저 자 소 개

김 직 수

- 2009년 5월 : 미국 메릴랜드 주립대학교 전산학과 (이학박사)
- 2009년 9월 ~ 2012년 5월 : 삼성 SDS CSP 연구소 수석연구원
- 2012년 6월 ~ 2017년 2월 : 한국과학기술정보연구원 선임연구원
- 2017년 3월 ~ 현재 : 명지대학교 컴퓨터공학과 조교수
- ORCID : https://orcid.org/0000-0002-0104-4617
- 주관심분야 : 빅데이터 플랫폼, 클라우드 컴퓨팅