한국빅데이터학회지
제4권 제2호, 2019, pp. 73-83

워게임 시뮬레이션 환경에 맞는 빅데이터 분석을 위한 분산처리기술

Distributed Data Processing for Bigdata Analysis in War Game Simulation Environment

배민수

서울대학교 응용공학과

요 약

4차 산업혁명의 기술 등장 이후 대규모 데이터 시대에서 새로운 가치 창출을 위한 데이터 정보 분석은 다양한 분야에서 시도되고 있다. 대용량 데이터를 빠르게 처리하는데 있어서 분산 데이터 처리는 이미 필수적이다. 하지만 아직 국방 분야에서 운용하고 있는 시뮬레이션들은 쌓여 있는 비정형 데이터를 활용할 수 있는 시스템이 미비하다. 이에 본 연구에서는 훈련간 발생하는 문제에 대응하기 위한 지휘결심에 가시화된 데이터를 제공하기 위해서 대대급 규모의 시뮬레이션 모델에 적용 가능한 분산 처리 플랫폼을 제안한다. 전략게임 데이터 50만개를 분석하는 과정으로, 데이터가 가지고 있는 여러 요인들 중 승리요인에 영향을 미치는 요소들을 분석할 수 있게 구현하였다. 결과적으로 상위 10%에 있는 팀들의 데이터를 분석하는 과정에서의 분산처리 사용한 결과를 측정 및 비교 하였다.

■ 중심어 : 구성 모의실험, 분산 데이터 처리, 빅데이터

Abstract

Since the emergence of the fourth industrial revolution, data analysis is being conducted in various fields. Distributed data processing has already become essential for the fast processing of large amounts of data. However, in the defense sector, simulation used cannot fully utilize the unstructured data which are prevailing at real environments. In this study, we propose a distributed data processing platform that can be applied to battalion level simulation models to provide visualized data for command decisions during training. 500,000 data points of strategic game were analyzed. Considering the winning factors in the data, variance processing was conducted to analyze the data for the top 10% teams. With the increase in the number of nodes, the model becomes scalable.

■ Key Words : Constructive Simulation, Distributed Data Processing, Bigdata

2019년 11월 28일 접수; 2019년 12월 10일 수정본 접수; 2019년 12월 31일 게재 확정.
† 교신저자 (zpflaksen9@snu.ac.kr)

Ⅰ. 서 론

대규모 데이터 시대에 정보 분석을 위한 분산 데이터 처리는 대용량의 데이터를 빠르게 처리하는데 있어 이미 필수적이다. 이제는 데이터의 양에 문제가 아니라 그 데이터를 활용하여 어떤 의미를 갖는 정보가 만들어지는지가 더 중요해졌다. 실제로 이를 위해 다양한 빅데이터 솔루션이 개발되고 운용되고 있다. 하지만 아직 군에서 운용하고 있는 구성 모의실험에는 데이터를 활용할 수 있는 시스템이 미흡하며, 훈련 종료 후 쌓여 있는 비정형 데이터들을 가지고 사후 분석하는 것이 현실이다. 텍스트나 이미지 등 다양한 형태의 데이터들을 포함할 수 있는데, 이런 데이터들이 체계적으로 활용된다면 그 활용도가 무궁무진하다.

군에서도 현재 이런 데이터 활용 문제를 극복하기 위해 다수의 시뮬레이션을 복제하여 하나의 프로그램으로 구성하고 이를 시뮬레이션화 하여 결과를 얻고자 하는 Adaptive distributed parallel Simulation environment for Interoperable and Models (AddSIM) 개발 등의 노력이 진행되고 있다. AddSIM의 경우, 분산 환경에서 병렬 처리가 가능한 구조로 발전하는 추세를 반영하고, 국내 환경에 적합한 분산 환경 지원과 병렬 처리가 가능하도록 아키텍처를 개발하였다. 하지만 이런 연구들은 다양한 무기체계를 동시에 활용하는 조건이 포함되기 때문에 창 끝 부대에서 활용하기에는 다소 제한적이다. 이에 본 연구는 시뮬레이션 모델에 적용 가능한 분산 처리 아키텍처를 제시하고, 실시간 데이터 분산 처리를 통해 데이터 분석을 할 수 있는 플랫폼을 제안하였다.

이 플랫폼을 이용한 데이터 분석을 통해 비정형 데이터들을 활용하여 새로운 가치를 만들어 낼 수 있다면 훈련 상황에 실시간으로 발생하는 데이터들을 분석하여 객관적인 근거를 제시할 수 있게 되며, 더 나은 지휘 결심을 하는데 영향을 미칠 수 있을 것이다. 이에 본 연구에서는 시뮬레이션 운용간 실시간으로 발생하는 데이터를 분석하기 위한 분산처리 플랫폼을 적용하여 축적된 데이터를 분석하고 그 실효성을 검증하고자 한다.

Ⅱ. 배경 지식과 관련 연구

2.1 구성 모의실험

시뮬레이션은 복잡한 문제나 현상들을 해석하고 해결하기 위해 실제 모형이나 현장을 만들어 모의적으로 실험하고 그 특성들을 파악하는 것이다. LVC (Live, Virtual, Constructive) 시뮬레이션은 M&S (Models and Simulation)을 분류하는데 사용되는 기준이다. 시뮬레이션을 실제 환경, 가상 환경 그리고 모의 환경으로 분류하는 것은 워낙 장비와 현실성 정도가 다양하다 보니 명확한 구분이 없지만, 미국 국방부에서는 다음과 같이 정의하고 있다.

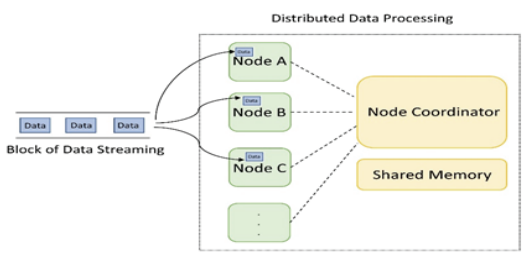
Live Simulation은 실제 사용하는 시스템이나 장비들을 실제 사람들이 직접 운용하는 방식으로 구성된 시뮬레이션을 의미한다. 실제 장비를 실제 사람들이 직접 운영하지만 살아있는 적을 상대로 수행하지 않기 때문에 이는 시뮬레이션으로 간주된다. Virtual Simulation은 실제 사람들이 모의로 구성된 시스템이나 장비를 운영하는 실험을 의미한다. 가상 시뮬레이션은 주로 제트기나 탱크 시뮬레이션으로 활용되는 모터 제어기술, 사고나 재난 등을 훈련하는 의사 결정 기술, 그리고 지휘・통제・통신・정보수집 등에서의 의사소통을 위한 기술 등이 있다. 예를 들면 파일럿이 비행 시뮬레이터를 통해 훈련하고 비행 연습하는 것을 말한다. Constrctive Simulation의 경우 주로 컴퓨터 프로그램으로 구성되어 있는 구성 모의실험은 가상의

사람들이 모의로 구성된 시스템이나 장비를

운영하는 시스템을 말한다. 시스템을 실험하는 실제 사람들은 입력 값을 주지만 결과를 결정하는 데는 관여를 할 수가 없다. 예를 들어 군 내에서의 시뮬레이션 사용자가 시뮬레이션 내의 유닛에게 적과 교전하도록 데이터를 입력할 수는 있지만, 적과의 교전 효과 및 발생할 수 있는 피해 등을 결정할 수 없는 것이다. 이런 시뮬레이션들은 다양한 유형의 모델이 있으며 Computer Generated Forces (CGF) 도 그 예제들 중 하나의 모델이다. 현재 운용되는 워게임 체계의 기본적인 구조는 동일한 네트워크 상에서 주 모델과 각 통제 및 운용반, 서버, 그리고 사후검토 모델이 연결되어 있으며 각 통제반에는 세부적으로 더 많은 운용자들이 존재한다.

2.2 분산 데이터 처리

과거 한 연구에 따르면 데이터 분산 처리는 시스템을 이루고 있는 물리적인 요소가 분산되었을 때 이루어졌다고 말한다. 물리적 구성요소로는 하드웨어와 처리논리회로, 데이터 그리고 운영체제를 의미하며, 이러한 구성 요소 중 하나가 여러 개의 시스템으로 나뉘어 각각의 객체를 처리할 수 있게 연결되는 것을 의미한다[2]. 하지만 이런 정의는 시스템 상에서의 일부분의 구성요소에 대한 물리적인 분산처리만을 의미하여 새롭게 다시 정의될 필요가 있었다. 지금의 데이터 분산처리는 좀 더 네트워크상의 컴퓨터 배열이 중심이 된다. 데이터 처리 기능의 네트워크상에 분산되어 있고, 특정 작업은 해당 컴퓨터를 사용하는 사용자나 멀리 떨어져 있는 컴퓨터에 의해 수행될 수 있다. Fig. 1과 같이 이런 네트워크상의 배열은 물리적인 공간의 제한이 없기 때문에 확장성에 있어서 강점을 가지고 있으나 그만큼 더 많은 네트워크 관리 자원이 필요하게 된다.



〈Fig. 1〉 Distributed data processing[3]

처음 분산 데이터 처리를 설명함에 있어서 데이터 분산 처리란 데이터들을 처리하기 위해 데이터를 같은 네트워크 상의 그룹 내에서 실행중인 여러 개의 노드로 분산시켜 처리하는 것이다. 한 그룹안에 존재하는 모든 노드는 각자 할당된 태스크를 실행하고, 네트워크로 연동되어 있기 때문에 서로 연동하여 작동한다. 이런 프로세스들은 각자 병렬적으로 실행되나 논리적으로는 하나의 프로세스처럼 수행하도록 보인다. 데이터와 노드, 메모리 등의 시스템은 통신망을 통해 연결되어 있기 때문에 유용한 자원을 공유하여 사용할 수 있으며, 각 노드가 여러 개의 태스크를 동시에 처리하기 때문에 속도가 향상된다. 만약 여러 노드 중 하나의 노드가 오류를 발생시키더라도 다른 노드들은 지속적으로 태스크를 처리할 수 있기 때문에 신뢰성 측면에서도 효율적이며, 물리적으로는 각 시스템들이 멀리 떨어져 있을 수 있지만 통신망으로 구성되어 있기 때문에 공간 제한적인 면을 극복할 수 있다.

분산 데이터 처리에는 Batch Processing과 Real-time Processing 이 두 가지의 처리 유형이 있다. Batch Processing은 지금까지 사용되었던 기존의 데이터 처리방식으로 특정시간에 컴퓨터 프로그램의 흐름에 따라 순서대로 발생한 데이터들을 수집하여 한꺼번에 처리하는 방식이다. 개별적으로 처리 요청들을 한 번에 처리하기 때문에 대량의 데이터를 처리하게 되는데 이는 업무의 효율성을 높이고 시스템의 과부하를 줄였다. 데이터가 처리되는 시간은 스케줄상의 특정 시간에 예약되어 수행되거나, 규칙적인 간격으로 처

리가 실행되는 것이 아니면 임의로 시행되지만 실시간으로 처리되지는 않는다.

이에 반해 Real-time Processing은 데이터가 입력될 때 짧은 시간동안 데이터를 실행시켜 즉각적으로 데이터가 출력되는 방식이다. 데이터에 대한 정보를 얻기 위해 데이터 분석을 추가적으로 실행한다. Batch Processing이 프로그램의 흐름에 따른다면 Real-time Processing은 시간에 따른 변화라고 할 수 있다. 또한, 실시간으로 처리를 수행하는 동안 응답이 지연되거나 마스터 정보가 최신이 아닐 경우가 없기에 즉각적이 조치가 가능하고, 이렇게 업데이트가 된 최신 데이터를 가지고 사용자는 보다 더 나은 결론에 도출할 수 있다. 결국 Real-time Processing은 Batch Processing에 비용이 많이 들고 구조가 복잡하다는 단점이 있으나 현재 지속적으로 연구되고 활용되는 처리방식이다.

온라인 서버나 웹사이트 등 다양한 분야에서 분산 데이터 처리가 사용되고 있지만 지금도 지속 개발중인 이유는 구조가 복잡하기 때문이다. 여러 노드와 부수적인 통신 장비들이 요구되기 때문에 각 노드에 대한 관리 및 문제 발생시 해결하는 체계를 구성하는데 개발이 어려운 것이다. 또 다른 단점으로는 보안 문제가 발생한다. 인증되지 않은 장치가 분산 데이터 처리 구조에 연결된 경우 데이터나 장치 및 다른 성능 상에 영향을 미칠 수 있으며 데이터 손실도 가능하기 때문이다.

분산 데이터 처리가 시스템 유지상 통일성을 잃기 쉽긴 하지만 그럼에도 많은 장점들로 인해 활용도가 높다. 첫째, 중앙 집중식 처리에 비해 비용이 저렴하다. 중앙 집중식 처리를 하는 기업은 대규모 처리를 위해 메인 프레임과 슈퍼컴퓨터가 있어야한다. 그 비용부터 많이 들며 중앙 컴퓨터가 오류가 나는 경우 모든 데이터가 손실될 수 있는 위험도 가지고 있다. 반면 물리적으로 분리된 공간에서 다른 컴퓨터를 연결하여 처리하면 데이터 활용도 유용하고 쉽게 노드를 추가하거나 제거함에 있어서도 유연하기 때문에 메인 프레임의 과부하를 감소시킨다. 둘째, 확장성이 크고 하나의 일을 여러 시스템이 동시에 처리하기 때문에 속도 및 신뢰도가 향상된다. 계획된 데이터보다 많은 양의 처리가 필요한 상황이 발생하면 현재 구조에 컴퓨터를 추가하여 연결하면 쉽게 해결할 수 있다. 또한 병렬 처리로 인해 연결되어 있는 모든 노드의 데이터가 동시다발적으로 업데이트가 되어 컴퓨터를 추가하거나 제거하는 경우에도 데이터의 흐름에 영향을 미치지 않는다. 셋째, 여러 자원들이 연결되어 있기 때문에 사용자나 자원들 간의 통신이 용이하다. 물리적으로 분리되어 있어도 서로의 자원을 사용함에 있어서 제한이 없게 된다. 넷째, Fault-tolerance가 높다. 단일 시스템에서 장애가 발생하는 경우 처리가 지연되지만, 분산 데이터 처리 시스템의 경우 장애 발생시 다른 노드에게 요청하여 실행할 수 있기 때문에 안정성이 높다. 데이터의 경우에도 복제되어 서로 공유할 수 있기 때문에 장애로 인해 데이터에 접근할 수 없지 않고 오히려 중앙 집중식 시스템보다 빠른 속도로 데이터에 접근할 수 있는 장점이 있다.

Ⅲ. 분산 처리 시스템 모델

3.1 현존하는 시스템 모델 비교

데이터 처리를 하기 위해 가장 많이 이용되는 플랫폼은 Hadoop이 대표적이었다. 하지만 Map-Reduce가 작업이 많은 분산 작업 실행을 할 때 Disk를 통한 데이터 공유로부터 발생하는 성능 저하 단점이 발생했고, 이를 극복하기 위해 In-Memory 방식으로 메모리를 읽기 전용 방식으로 활용하는 Spark가 개발되었다. Spark는 인메모리 기반의 빅데이터 처리 플랫폼으로 분산 컴퓨터의 메모리에 저장되는 RDD와 이를 활용하기 위한 인터페이스를 제공한다. Table 1은 이러한 대표적인 플랫폼

〈Table 1〉 Hadoop, Spark, Ray performance comparison

구 분	Hadoop	Spark	Ray
Speed	Fast	100x faster than MapReduce	1 million task per second
Data Proccesing	Batch processing	Batch processing	Real-time processing
Store	Store Data on Disk	Store Data Memory	Store Data Memory
Written in	Java	Scala	Python
Framework extensibility	x	x	o
Latency	High	Medium (Second)	Low(Micro-Second)

Hadoop, Spark와 본 연구에서 활용한 플랫폼을 비교 분석한다.

개발언어 관점으로 보면 Hadoop과 Spark는 Java로 개발되었고, Ray는 Python 언어로 개발되었다. Java에 비해 Python은 훨씬 사용하기 쉬운 스크립트 언어로써, 작성할 때 Interpreter가 변수 유형을 유추하고 실행할 때 체크하기 때문에 최초에 변수 타입을 설정할 필요가 없다. 영어와 유사하게 사용되는 구문이 많아서 누구나 쉽게 코드를 사용할 수 있다. 이에 반해 Java는 엄격한 Syntax 규칙을 따라야한다. Dynamically typed 언어인 Python에 비해 Statically typed의 언어로써 변수 타입을 선언하지 않으면 코드가 컴파일 되지 않는다. Java나 Python 둘 다 유용한 언어이며 개발자들은 현재 Java를 더 많이 사용하긴 하지만 차후 설명한 Framework 확장성에서 누구나 쉽게 접근할 수 있기 위해 본 연구에서는 Python으로 개발된 Ray를 사용하였다.

데이터 처리방식의 관점에서 2절에서 설명했듯이 Batch Processing은 대량의 데이터를 처리하는 효율적인 방법이다. 시간의 분 단위 이상이 대기시간으로 생성되지만 일괄적으로 대량의 데이터를 처리하는데 이상적이다. 하지만 본 연구에서 Real-time Processing이 필요한 이유는 시뮬레이션이 진행되는 도중에 필요한 분석 결과를 빠르게 얻기 위해서이다. Real-time Processing을 통해 얻게 되는 정보가 항상 최신 상태를 유지할 수 있기 때문에 즉각적인 조치를 취할 수 있게 된다. Spark의 경우 Spark streaming 같이 Stream Processing을 접목시키고 관련된 API들이 등장하였지만 모든 부분을 다루기에는 광범위하여 본 연구에서는 플랫폼의 기본 처리과정에 대해서만 비교한다.

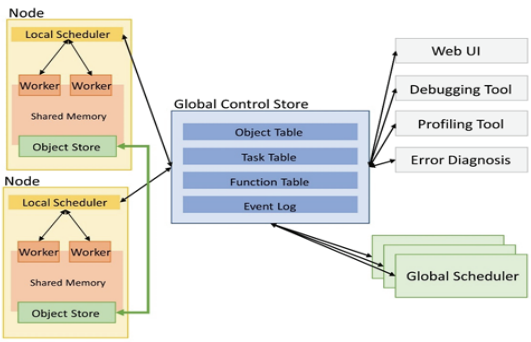
Ray는 Block synchronous 패러다임을 지양한다. 태스크가 수행될 때 그에 종속되는 모든 작업들이 실행될 준비가 되어있는 Just-In-Time의 데이터 흐름의 아키텍처가 필요하고, Ray는 Hadoop과 Spark의 기능에 이러한 유용성을 갖추고 있다. 노드의 수에 영향을 받지만 초당 백만개의 태스크를 처리할 수 있는 속도를 갖고 Latency에서 우위를 갖기에 효과적인 수단이다.

3.2 절 시스템 아키텍처

Ray는 병렬 및 분산 파이썬을 위한 범용 프레임 워크로, 스케줄링 된 태스크와 작업 실행을 담당하는 실행 엔진이다[1]. 기존에도 Pytorch나 Spark 등 이와 비슷한 기능을 가지고 있는 프로그램들은 존재하지만 AI와 빅데이터들의 다양한 워크로드의 발생들로 인해 여러가지 요구사항을 만족시키지 못했다. Python 언어 내에서 활용할 수 있는 멀티 프로세싱 모듈이 있지만 이 역시 최신 응용 어플리케이션들의 다음의 요구사항을 만족하지 못한다.

- 하나 이상의 머신에서 같은 코드 수행 여부
- State를 가지고 Communication이 가능한 actor와 microservice 구축
- 적절한 Fault tolerance
- 대규모 데이터나 수치 데이터의 효율적 처리

Ray는 이러한 요구사항들을 만족시키며, 간단한 작업은 더욱 간단하게 하고, 복잡한 작업들은 가능하게 만들었다. 또한 Open MPI와 같은 Low level primitive와 Spark나 TensorFlow같은 High level abstraction의 경우, 서로 다른 Abstraction을 제공하기 때문에 Application을 다시 작성해야 한다. 하지만 Ray의 경우 기존에 사용했던 Function와 Class의 개념을 Task와 Actor로 분산 세팅하여, 큰 수정없이 Application을 Parallelized 할 수 있다.



〈Fig. 2〉 System model hierarchy[4]

Fig. 2에서 보이는 바와 같이 시스템은 두 가지의 계층으로 구성된다. App layer에서는 Driver와 Worker 그리고 Actor로 구성된다. Driver는 사용자 수준에서 사용자가 실행하는 프로그램의 프로세스이다. Worker는 시스템 상에 있는 태스크나 함수들을 실행하는데, remote function으로 실행된 태스크가 포함된 Stateless한 프로세스로써 드라이버나 다른 Work들에 의해 생성된다. remote function이 선언되면 기본적으로 Worker에게 수행이 주어지는데, 이외에도 드라이버나 스케줄러가 태스크가 생성되면 default로 Worker에게 수행을 요청한다. Stateful한 프로세스를 수행하는 Actor는 호출될 때만 보이며 사전에 정의된 메서드를 수행한다. Actor는 Work와 Driver에 의해 일시적으로만 수행되며 Worker와 동일하게 메서드를 순차적으로 실행한다. 하지만 Actor는 Stateful한 프로세스를 수행하기 때문에 이전 메서드 실행으로 인한 결과에 의존하여 연속적으로 수행한다.

App layer의 요소를 제외한 나머지 구성요소들은 System layer에 속한다. System layer의 주요 기능을 하는 요소는 Global Control Store 와 Distributed Scheduler 그리고 Distributed Object Store이 있다. Global Control Store는 시스템의 전체적인 제어를 실행하며 하나의 키 값 저장소라고 볼 수 있다. Pub-sub 기능을 가지고 있는데 이는 하나의 채널을 오픈 시켜 놓고 여러 개의 프로그램이 오픈 되어 있는 채널을 구독(Sub)하게 한 뒤, 하나의 프로그램에서 이벤트가 발생되면 그 이벤트를 모든 구독한 프로그램들이 받을 수 있도록 발행(Pub)하는 기능이다. Global Control Store은 초당 수백만개의 태스크를 동적으로 생성할 수 있는 시스템의 내결함성과 낮은 Latency을 유지하기 위해 설계되었다. 동시에 낮은 Latency를 위해서는 스케줄링에서도 오버 헤드를 최소화해야 한다. 이를 위해 각 Object 전송에 스케줄러를 포함시켜, Object의 메타데이터를 Global Control Store에 저장하여 태스크 디스 패치와 스케줄링을 분리시킨다.

remote function을 정의했을 때 수행하는 과정은 먼저 정의된 remote function을 Global Control Store에 저장하고, 시스템상의 모든 Worker에게 공유된다. Driver는 함수를 Local Scheduler에게 보내고 이 함수는 최종적으로 Global Scheduler에 입력된다. Global Scheduler는 Global Control Store에서 해당 함수의 위치를 전달받고 태스크를 수행할 노드를 선정한다. 선정된 노드의 Local Scheduler는 Local Object Store에 해당 함수가 저장되어 있는지 확인하고 없는 경우에는 Global Control Store에서 해당 함수의 위치를 확인하는 과정을 수행한다. 함수가 수행된 return값들은 Global Object Store에 저장된다.

Ⅳ. 실험 측정 및 결과분석

4.1 자료 분석

군의 구성 모의실험은 작전 계획을 토대로 전략 게임을 실행한다. 실제 비문을 바탕으로 구성되는 전략 게임이기 때문에, 이와 비슷한 전략 시뮬레이션의 데이터가 필요하다. 과학화 전투훈련(KCTC)에서 1개의 훈련 부대가 2주간 주야 연속으로 공방 전투를 실시하면 훈련 간 생성되는 데이터는 교전, 음성, 그리고 영상 정보가 약 300GB정도 발생한다. 실질적으로 구성된 환경에서 전투경험을 익히고 전장에서의 기술이 기록되는데, 특히 차량을 통한 이동 비디오 영상 정보나 전술통신장비들을 통한 감청 정보, 전투 간 발생하는 교전정보들이 주를 이룬다. 정량적으로 표현할 수 있는 데이터 중 승리 요인에 영향을 미칠 수 있는 데이터는 먼저 적의 강점, 약점을 탐지할 수 있는 주 노력 방향이나 부대 규모 등이 있다. 또한 적 방어체계에 대한 기동, 화력, 정보 체계나 기습을 위한 시간, 장소, 수단에 대한 데이터도 있다. 이러한 정보들 중 공격 작전의 승리 요인에 상대적으로 보다 더 많은 영향을 미치는 독립 변수들은 침투 부대 운용 규모, 공격개시선 통과 및 진출 속도, 지휘관의 생존 등이 있다[5]. 결국 시뮬레이션 데이터에서 이런 정보들을 추출해 분석해야 한다는 의미를 갖는다.

이에 본 연구에서는 팀으로 구성된 플레이어들이 전략을 가지고 여러 가지 다른 화기 종류를 이용하여 생존해 나가는 전략 게임 데이터를 활용하였다. 데이터는 50만명 플레이어의 데이터를 사용하였으며 데이터가 포함하고 있는 요소는 팀 규모, 화기·구급품·방탄용품 등의 자산, 차량으로 이동한 거리, 도보로 이동한 거리, 교전을 통해 사망한 인원 수, 팀별 생존 시간, 그리고 팀 위치이다. 이 요소들은 승리요인에 영향을 미칠 수 있는 항목들로 선별하였으며 이를 통해 다음을 분석하고자 한다.

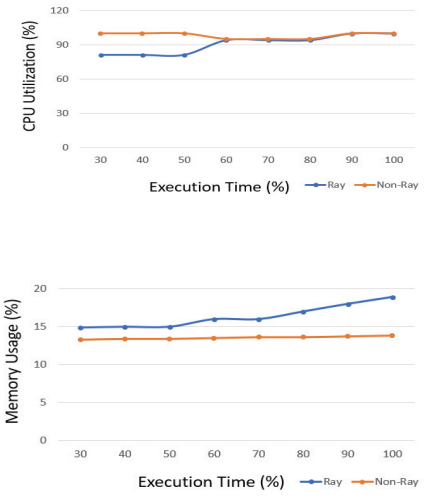
- 팀별 교전 시 승리 한 횟수
- 교전에서 최종 승리한 상위 10%팀의 데이터 분석
- 팀 단위별 받은 피해량

상위 10%의 팀 데이터 분석의 경우 생존 플레이어를 통해 지휘관의 생존시간과 전투력 집중을 위해 구성원들의 거리를 확인할 수 있다. 구성원들의 거리 간격을 줄이고 지휘관의 생존 시간이 길수록 해당 팀의 생존성은 증가할 수 있다[6]. 따라서 상위 팀의 다수의 승리 기록과 이에 영향을 미친 요인을 분석하게 되면 차후 우리나라에 맞는 각종 교리를 검증할 수도 있고 또한, 국방 M&S 체계의 모의 논리를 보강할 수 있는 역할도 가능할 것이다. 지형 및 전쟁 환경에 대한 전투원들의 행동패턴을 평가하고 생존성을 보장할 수 있는 요인을 파악한다면 현재 군 전투력을 위한 교육훈련체계에 발전방향을 제시하게 되어 나아가 조직의 생산성이 증가할 것이며 국방력 증진에 영향을 미칠 수 있을 것이다

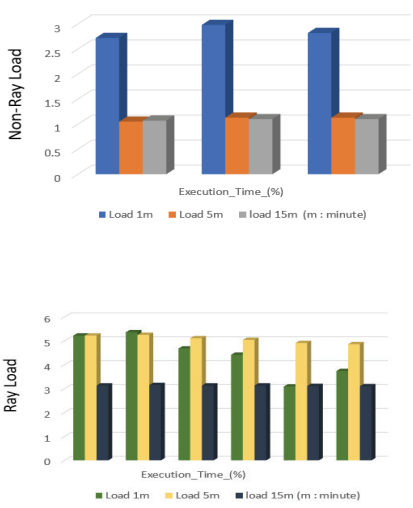
4.2 결과 분석

첫번째 실험에서는 데이터 분산처리 플랫폼 사용 유무를 비교하여 CPU Utilization의 상관관계를 도출하는 것을 목적으로 하였다. 이어서 단일 노드 경우의 Memory 사용률을 비교 분석한다. 또한 전체 프로세스 수행 시간과 Load 값의 변화를 비교하여 실제 분산 데이터 처리 플랫폼에 따른 시간당 Latency와 유용성에 대하여 평가하였다. 먼저 Fig. 3과 Fig. 4에서의 결과는 지속시간에 따른 CPU 이용률과 Memory, Load 변화를 보여준다. 이 그래프에서의 행의 지점은 두 실험의 총 수행시간이 다르기 때문에 총 수행시간에 따른 평균 시간 비율로 설정하였다. 분석 시스

템 모델을 사용하는 CPU 사용률은 사용하지 않는 경우와 비슷하지만, 사용하지 않는 경우가 더 높은 결과를 보인다. 전체적으로 분산 처리 시스템을 사용할 경우 전체 프로세스 수행시간의 50% 이하 시간에서 1~18.7% 정도의 자원 사용률



〈Fig. 3〉 프로세스 수행시간에 따른 CPU Utilization과 Memory Usage 변화량



〈Fig. 4〉 프로세스 수행시간에 따른 Load 변화

감소를 보였다.

Memory 사용률에서는 분산 데이터 처리 모델을 사용하는 경우 Object의 메타 데이터를 Redis 서버에 저장하는 과정에서 메모리 사용률이 증가한다. 데이터를 서버에 저장하는 시작부터 플랫폼을 사용하지 않은 경우보다 사용량이 많은 것을 볼 수 있다. 그래프의 행이 의미하는 Execution time은 전체 작업의 수행 시간을 의미하며, 이전의 실험과 동일하게 경우마다 작업 수행시간이 다르기 때문에 총 수행 시간에 따른 평균 시간 비율로 나타냈다. 그래프의 열이 의미하는 바는 시스템에서 사용 가능한 메모리의 양에 비해 현재 사용중인 메모리 값의 비율이다. 실험 설정의 노드의 수가 적기 때문에 분산 처리 플랫폼을 사용했을 경우가 높게 나왔지만 노드 수를 증가시키거나 재생 버퍼 사이즈를 줄임으로써 사용률을 감소시킬 수 있다. 본 실험에서는 적은 노드의 수로 50만개의 데이터를 처리했을 때의 결과를 다루었기에 발생한 결과이다. 적은 노드에서 데이터의 수가 50만개 이상의 경우에는 메모리 이슈가 발생했는데 이는 향후 연구과제로 선정하고자 한다.

Distributed scheduler에 의해서 태스크가 수행하는 동안 태스크 상태가 지속적으로 변하는데 이 State 중 실행하는 동안을 Load로 측정한다. 현재 수행중인 태스크와 수행 대기하고 있는 태스크의 수치를 주기적으로 계산하여 1분, 5분, 15분 기준으로 평균치를 계산한다. 결과 그래프에서 행은 분산 처리 플랫폼을 사용하지 않았을 때 전체 작업 수행 시간을 비율로 표시한다. 작업 수행이 시작했을 당시 0%으로 시작하여 모든 작업이 종료되었을 때에는 100%로 나타낸다. 전체 작업의 수행 시간 중 1분, 5분, 15분 평균으로 나타나는 Load값을 막대그래프로 나타냈는데, 값이 변화할 때를 기준으로 나타난다. 그래프이 세로축은 측정되는 Load값이다. 분산 처리를 수행하지 않았을 때에 1분 평균 로드는 2.84, 5분 평균

로드는 1.103, 15분 평균 로드는 1.09로 측정되었다. 수행시간이 증가할수록 처리하는 태스크의 양이 감소하기 때문에 전체적인 로드 값이 감소한다.

플랫폼을 활용하여 수행한 경우에도 수행시간이 증가할수록 전체적인 태스크의 수는 수행으로 인한 감소가 이루어지기 때문에 하향세를 보인다. 분산 처리를 수행했을 때에는 1분 평균 로드 4.418, 5분 평균 5.065, 15분 평균 3.125가 측정되었다. 그래프에서의 가로축은 위의 실험과 동일하게 전체 작업의 수행 시간을 비율로 표기하였고, 세로축은 Load값을 나타낸다. Load값은 CPU에 의해 실행 중이거나 대기 상태에 있는 프로세스들의 주어진 1분, 5분, 15분 동안 계산된 평균 시스템 Load값이다. 본 연구에서의 Load값들이 실험 환경의 CPU Core Thread 수보다 낮은 수로 측정되어 성능상의 문제는 없는 것으로 보이나, Non-Ray로 수행했을 때와는 평균 Load값은 전반적으로 큰 수치가 측정되었다. 이는 remote function과 발생하는 Actor들로 인해 수행되는 프로세스가 증가하기 때문으로 보인다. 단연 CPU 코어 수나 노드가 증가할수록 각 노드의 에서의 다른 프로세스들을 동시에 처리될 수 있기 때문에 CPU 코어 수보다 높지 않은 수치는 멀티 프로세싱이 가능하다. 결국 코어나 노드의 수에 따라 Actor를 변화시켜 최대 수행 가능하게 설정하여 수행시킬 수 있다는 것을 확인하였다.

전체 프로세스 수행 시간을 비교하면, 분산 데이터 처리를 수행했을 때 Redis 서버에 접속하는 시간을 제외한 처리 수행 시간이 더 짧은 것을 확인할 수 있었다. 본 실험에서의 데이터 분석이 비교적 간단한 처리 수행이기 때문에 분산 처리를 위한 서버 접속의 시간이 영향을 미치는 결과가 확인되었다. 다만 어느 정도 크기의 데이터와 분석 규모에 몇 배 이상의 향상을 보이기 위해 몇 개의 노드가 기본적으로 필요로 요구되는지에 대한 각각의 기준점을 잡기에는 어려움이 발생하였다. 이는 실제 전략 데이터를 활용하여 필요한 정량화 된 데이터의 기준이 우선적으로 선정된 이후에 데이터 처리로써 향후 연구 방안을 정리하고자 한다

Ⅴ. 결 론

현재까지 국방분야에서 데이터 분석에 관한 연구는 그 필요성을 중요하게 언급할 뿐 분석 기술이나 데이터 활용에 대한 연구는 부족한 실정이었다. 이전 연구에서는 사회 공공이나 민간부문에서 빅데이터로 다양한 정보를 추출해 유용한 통찰력과 지식을 얻는 사례와 미군이 국방력을 증진하기 위해 활용하는 아키텍처들을 통하여 우리 군에서도 새로운 가치창출을 위한 노력이 필요하다는 고찰이 있었다[7]. 각종 훈련 체계 간의 연계성이 부족하여 데이터의 활용에 제한 사항을 극복하고자 서론에서 기술했던 Adaptive distributed parallel Simulation environment for Interoperable and Models (AddSIM) 등을 활용해 LVC 기반의 과학화 훈련 체계를 구축해 가고자 하는 연구 또한 진행되고 있다. 하지만 해당 연구의 범위가 방대하여 창 끝 부대나 현재 운용하고 있는 시뮬레이션에서의 데이터를 활용하기 위해 쉽게 접목시킬 수 있는 분산 데이터 처리 플랫폼을 제시하였다.

Ray는 병렬 및 분산 Python을 위한 범용 프레임워크로 remote function 함수로 태스크를 생성하고 분산 스케줄링을 실행하여 작업을 수행하는 실행엔진이다. 최초 remote function으로 정의된 함수는 Global Control Store의 Function table에 저장이 되고 이는 모든 Worker들에게 전달된다. Object들은 각 노드와 Global Control Store의 Object table에 저장이 되며, 드라이버는 id값을 로컬 스케줄러에게 전달한다. Global Control Store의 Object table로 인해 Local Object Store은 id값을 공유할 수 있다. 이러한 수행으로 분산 처

리를 하며 노드가 증가할수록 확장성을 갖는다.

본 연구의 실험에서 군사적 의미를 갖는 전략 게임 데이터를 가지고 위의 분산 데이터 처리를 수행하여 전체 프로세스 수행 시간 중 50%의 수행 시간 동안 1~18.7%의 CPU 자원 사용률 감소를 통해 효율성을 확인하였다. Memory 사용률에서는 분산 데이터 처리 모델을 수행하는 경우 Object의 메타 데이터를 Redis 서버에 저장하는 과정에서 메모리 사용률이 증가한다. 하지만 이는 노드 수의 증가나 재생 버퍼 사이즈를 줄임으로써 감소시킬 수 있다. Load의 평균값을 측정하였을 때 Actor와 Worker로 인해 생성되는 Process 증가로 인해 Load값이 증가하고 이는 멀티 프로세싱으로 처리량을 증가시킬 수 있음을 확인하였으며 코어의 수를 초과하지 않은 범위에서 노드 수를 구성하여 효율적으로 수행 가능하다.

결과 분석을 통해 발생한 메모리 이슈와 분산 데이터 처리를 위한 자원의 기준점은 향후 연구 방안으로 정리하고자 한다. 본 연구에서 적은 노드 개수로 50만개의 데이터 처리를 수행하였을 때 Object가 Global Control Store에 저장되는 과정에서 데이터의 크기로 인해 메모리 사용률이 증가하였으며 이는 메모리 이슈로 발전될 수 있다. remote function으로 정의한 함수가 대량의 array를 사용하거나 Object로 인한 원인이기 때문에 remote function에 의한 array와 데이터 크기에 따라 최대 효율을 실행시킬 수 있는 코어와 노드의 기준점을 설정해야 한다. 메모리 이슈와 기준점에 대한 연구가 추가적으로 진행되어야 하지만 해당 부분은 향후 연구 과제로 선정하고자 한다. 결과적으로 제시된 분산 데이터 처리 플랫폼으로 국방 데이터를 활용하여 가시화된 자료로 새로운 가치 창출을 할 수 있다

참 고 문 헌

[1] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. Jordan, and L. Stoica, “Ray: A Distributed Framework for Emerging AI Applications”, *USENIX Symposium on Operation Systems Design and Implementation*, Vol. 13, pp. 562-563, October 2018.

[2] P.H. Enslow, “What is a “Distributed” Data Processing System?”, *Computer*, Vol. 11, pp.13-14, January 1978.

[3] “Distributed Systems - Distributed Data Processing 101”, www.8bitmen.com.

[4] C. Sarthak, “Scaling Python Modules Using Ray Framework”, https://medium.com. Accessed May 28, 2018.

[5] G.G. Kim, D.S. Kim, “Development and Application of Effect Measurement Tool for Victory Factor in Offensive Operations Using Big Data Analytics”, *The Korean Operations Research and Management Science Society*, Vol. 39, No. 2, pp. 124-125, June 2014.

[6] J.O. Kim, H.J. Cho, G.G. Kim, “Analysis of Survivability for Combatants during Offensive Operations at the Tactical Level”, *The Korean Journal of Applied Statistics*, pp. 929-930, 2015

[7] S.W Kim, G.G. Kim, B.K. Yoon, “A Study on a Way to Utilize Big Data Analytics in the Defense Area”, *The Korean Operations Research and Management Science Society*, Vol. 39, No. 2, pp. 2-4, June 2014.

저 자 소 개

배 민 수(Minsu Bae)

·2013년 : 숙명여자대학교 수학통계학부(학사)

·2013년 ~ 현재 : 대한민국 육군 장교

·2018년 ~ 현재 : 서울대학교 응용공학과(석사)

·관심분야 : 빅데이터 분석, 데이터 관리, 운영체제