Spark Streaming 기반 시스템에서 실시간 처리 지원을 위한 선별적 자원 할당 기법

이은성[1)○], 김정호[2)], 홍성수[1), 2)]
[1)]서울대학교 전기·정보공학부
[2)]서울대학교 융합과학기술대학원 융합과학부
{eslee, jhkim, sshong}@redwood.snu.ac.kr

Selective Resource Managing Technique for Real-Time Processing in Spark Streaming System

Eunseong Lee[1)○], Jungho Kim[2)], Seongsoo Hong[1),2)]
[1)] Department of Electrical and Computer Engineering, SNU
[2)] Department of Transdisciplinary Studies, GSCST, SNU

요 약

빅데이터 기술의 발전과 함께 클라우드 기반 자동차의 실현이 가속화되고 있다. 이를 위해 클라우드 시스템은 복잡하고 어려운 실시간 요구사항들을 만족해야 한다. 그 중에서도 실시간 상태 유지 연산은 최우선적으로 만족해야 하는 실시간 요구사항들 중 하나이다. 현존하는 기술 중에서 해당 요구사항을 가장 잘 만족시키는 것은 Spark Streaming 프레임워크이다. 하지만 해당 프레임워크는 실시간성 보장을 위한 별도의 분산 자원 관리가 필요하다. 하지만 현존하는 분산 자원 관리 기술은 real-time semantics를 고려하지 않고 있다. 현존 기술은 비실시간 응용에 의해 실시간 응용의 자원 사용이 제한되는 상황을 방지하지 못한다. 본 논문에서는 이러한 문제를 해결하기 위해 선별적 자원 할당 기법을 제안한다. 이 기법은 런타임에 실시간 태스크를 식별하고 요청한 자원을 바탕으로 실시간 태스크가 비실시간 태스크로부터 자원 독점하는 것을 보장한다. 이를 Spark Streaming 2.1.0에 적용하고 실험을 하여 평균 수행 지연이 약 52.98% 감소함을 확인하였다.

1. 서 론

최근 ICT 기술이 자동차에 융합되면서 클라우드 기반 자동차에 대한 연구가 활발히 진행되고 있다. 이러한 자율 주행 자동차의 실시간 서비스 제공을 위해 클라우드 시스템은 복잡하고 어려운 실시간 요구사항들을 만족해야 한다. 클라우드 시스템이 최우선적으로 만족시켜야 하는 실시간 요구사항들 중 하나는 실시간 상태 유지 연산(real-time stateful operation)을 지원하는 것이다. 본 논문에서는 현존하는 기술 중 가장 낮은 지연의 상태 유지 연산을 지원하는 Spark Streaming 시스템을 대상시스템으로 선정하였다. 하지만 Spark Streaming 시스템에서 실시간성을 보장하기 위해서는 분산 자원 관리 계층이 필요하다.

이를 위한 현존 기술들 중 대표적인 것은 YARN과 MESOS가 있다. 현존 기술들은 자원 할당 정책에 따라 정적 할당과 동적 할당으로 나뉠 수 있다[1][2][3]. 정적 할당이란 응용이 요청한 자원을 고정적으로 할당받아 사용하도록 하는 정책이다. 하지만 이는 수행되는 응용이 많아질수록 보장 가능한 자원의 양은 비례하여 줄어들어 확장성이 떨어지는 문제가 있다. 동적 할당은 응용의 수행 중 다른 응용이 할당 받은 자원을 사용하지 않을 경우, 이를 사용할 수 있도록 런타임에 자원 할당량을 수정하는 정책이다. 하지만 해당 정책은 real-time semantics를 전혀 고려하지 않은 상태로 응용에게 자원을 할당한다. 즉, 자원을 동적으로 할당받는 비실시간 응용에 의해 실시간 응용은 자원 사용에 제한을 받을 수 있다.

본 논문에서는 Spark Streaming 시스템에서 위의 한계점을 극복하기 위해 선별적 자원 할당 기법을 제안한다. 해당 기법은 real-time semantics를 고려하여 실시간 응용이 사용 가능한 자원을 독점적으로 사용하도록 보장한다. 이를 위해 두 가지 작업들을 수행한다. 먼저 실시간 응용을 식별하고 그에 대한 정보를 자원 관리자에게 전달한다. 그리고 자원 관리자는 그 식별 정보를 이용하여 실시간 응용의 수행기간 동안 그 응용이 필요로 하는 자원 사용을 보장한다. 이를 통해 실시간 응용이 비실시간 응용과 자원 경쟁을 하면서 발생하는 지연을 줄인다. 제안된 기법은 Apache Spark Streaming 2.1.0이 탑재된 서버에 구현하였다. 대상 시스템에 제안 기법을 적용한 결과 실시간 응용의 평균 수행 지연이 약 52.98% 감소함을 확인하였다.

2. Spark Streaming 시스템의 동작

이 장에서는 본 논문의 이해를 돕기 위해 spark streaming에서의 잡(Job) 수행 과정과 잡이 사용할

자원에 대한 관리 기술에 대해 설명한다. 먼저 Spark Streaming 시스템에서 잡의 수행은 DStream을 입력 데이터로 받아 처리된다. DStream은 입력 받은 스트림 데이터를 일정 시간 간격으로 나눈 데이터이며, 이는 Spark의 RDD (resilient distributed dataset)라는 인메모리(in-memory) 자료 구조로 관리된다.

그림 1는 Spark Streaming 시스템의 계층적 구조를 나타낸다. Data analytics framework 계층에는 spark streaming 프레임워크가 존재한다. 해당 프레임워크의 핵심적 기능 중 하나는 Spark Streaming 시스템 위에서 동작하는 잡을 논리적 수행 순서를 갖는 태스크의 집합으로 변환하고 이를 실제 수행할 워커(worker)에 배분하는 것이다. 여기서 이 과정을 태스크 스케줄링이라 부르겠다. 이외에도 메모리 관리, 고장 복구 등의 기능을 제공한다.

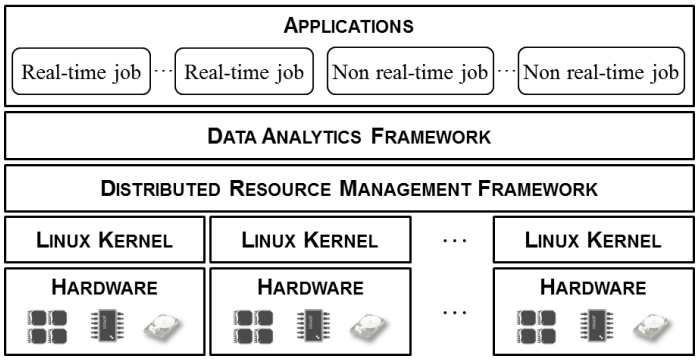

그림 1. Spark Streaming 시스템의 계층적 구조

그림 2는 Spark Streaming 시스템에서의 태스크 스케줄링 과정을 구조적으로 나타낸 예시이다. 먼저 마스터(master)는 수행할 응용의 잡을 스테이지(stage) 단위로 구분되고 잡을 구성하는 연산의 논리적 수행 순서를 나타내는 DAG (Directed Acyclic Graph)를 만든다. 스테이지는 같은 함수를 수행하는 독립적인 태스크들에 의해 수행된다. [4] 위와 같이 스테이지 단위로 구분된 태스크들의 메타데이터는 데이터의 지역성(data locality)을 고려하여 그 태스크가 처리할 파티션을 가지고 있는 워커에게 보내진다. 마스터는 워커에 존재하는 집행자(executor)에게 특정 태스크의 수행을 요청하고, 그에 따라 집행자는 워커에서 해당 태스크를 실제 수행시킨다.

Spark Streaming에서는 distributed resource management framework로 MESOS나 YARN을 사용할 수 있다. 해당 프레임워크의 주된 목적은 잡이 필요로 하는 자원을 손쉽게 확보하여 시스템이 안정적인 서비스를 제공하게 하는 것이다. 이들은 잡이 요청한 자원에 대한 정보를 상위 계층으로부터 받아 해당 태스크에 대해 요청 자원 사용률을 보장해준다. 잡은 CPU 코어 개수와 사용할 메모리 공간의 크기를 요청할 수 있다. MESOS는 리눅스의 자원 사용률 조정 기법인 cgroups를 사용하여 잡의 자원 사용률을 보장한다.

cgroups는 리눅스 시스템에서 동작하는 태스크의 자원사용률을 서브시스템(subsystem) 단위로 제어하는 기술이다. 서브시스템은 CPU, 메모리, 네트워크 등과 같이 시스템 자원 할당, 우선순위 지정, 모니터링등의 제어가 가능한 모듈이다. cgroups는 가중치(weight), 한계치(limit) 등의 매개변수를 통해 서브시스템의 사용률을 조정한다.

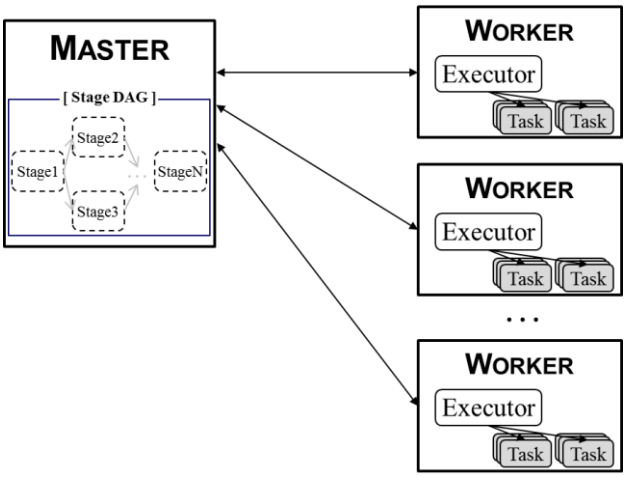

그림 2. Spark Streaming의 태스크 스케줄링 예시

3. 실시간 응용에 대한 선별적 자원 사용률 조정 기법

제안하는 기법은 Spark Streaming 시스템 위에서 동작하는 잡 간에 자원 경쟁을 관리하여 비실시간 응용과의 자원 경쟁으로 인한 실시간 응용의 지연을 줄이는 것을 목적으로 한다. 해당 기법은 런타임에 Spark 프레임워크로부터 실시간 잡을 구성하는 태스크들의 정보를 전달받고 이들이 필요로 하는 자원에 대한 추정치를 계산한다. 이를 통해 잡이 수행을 마칠 때까지 가용 가능한 자원을 독점적으로 사용하는 것을 보장한다.

그림 3은 제안 기법의 동작 개관을 시스템의 계층적 소프트웨어 스택 상에서 나타낸다. 제안 기법은 크게 실시간 태스크 그룹 식별자와 자원 관리자로 구성된다. 실시간 태스크 식별자는 실시간 잡의 요청 자원에 대한 사용을 보장할 대상 태스크 그룹을 식별하는 역할을 한다. 이를 위해 Spark Streaming 프레임워크로부터 해당 태스크 그룹의 이름을 입력으로 받아 각 태스크들의 ID를 대상 태스크 그룹 관리자에게 전달한다. 대상 태스크 그룹 관리자는 자원 사용률을 상향 조절할 태스크들의 ID와 요청한 자원에 대한 정보를 담은 테이블 입력으로 받아 요청하는 자원량에 따라 자원 사용의 우선순위를 정한다. 그리고 자원관리자에게 자원 사용률을 조정할 태스크의 ID와 자원 사용의 우선순위를 전달한다. 자원관리자의 역할은 CPU와 메모리 자원에 대하여 자원 사용률을 상향

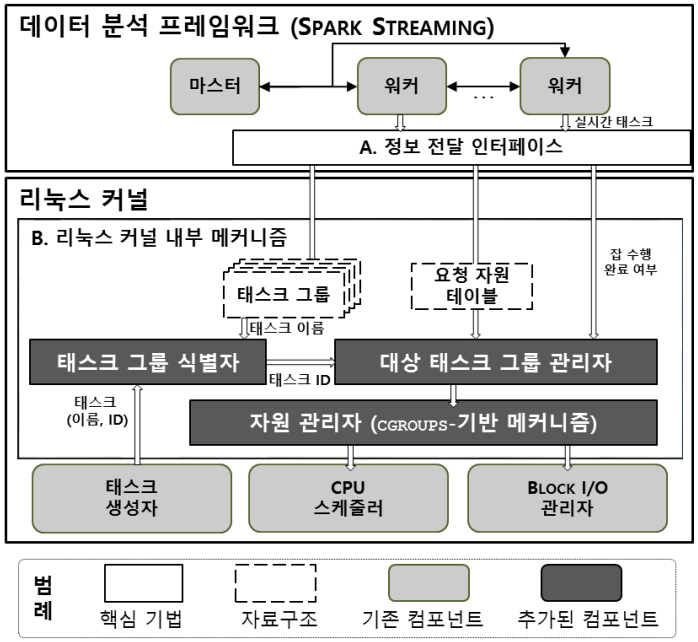

그림 3. 제안된 기법의 동작 개관

조정하고 비실시간 태스크들의 자원 사용률을 하양 조정하는 것이다. 잡의 수행이 끝나면 Spark Streaming 프레임워크는 수행 완료 여부를 대상 태스크 그룹 관리자에게 전달한다. 해당 정보는 자원관리자에게 전달되어, 비실시간 태스크들의 자원 사용률이 이전 상태로 복구된다.

위와 같은 방법으로 실시간 잡과 비실시간 잡 간의 자원 경쟁에서 실시간 잡에 우선권을 보장한다. 하지만 실시간 잡 간의 자원 경쟁이 발생한 경우를 대비한 공정한 자원 분배 정책이 필요하다. 제안 기법은 우선순위가 같은 실시간 잡들에게 하위 cgroups을 동적으로 생성해주어 위의 문제를 해결한다.

4. 실험 환경 및 검증 결과

제안된 기법은 다섯 대의 컴퓨팅 서버들로 구성된 실험환경에서 Spark Streaming과 리눅스 커널 내부에 구현되었다. 구체적인 실험 환경은 표 1과 같다.

검증 시나리오는 자율주행 자동차를 지원하기 위한 클라우드에서 실시간 맵데이터 갱신 작업이 비실시간 작업인 클라우드 기반 네비게이션, social media, 음악 스트리밍 작업과 함께 자원을 경쟁하는 것이다. 제안된

표 1. 실험 환경

하드웨어		
컴퓨팅 서버 5대	CPU	Intel i7-4790 3.60GHz octa-core
	메모리	DDR3 8GB * 4
	스토리지	1TB (SSD * 1)
네트워크	토폴로지	Fully Connected Network
	대역폭	1Gb
소프트웨어		
데이터 분석 프레임워크		Spark Streaming 2.1.0
분산 파일 시스템		Hadoop 2.7.2
운영체제		Ubuntu 16.04.1 LTS

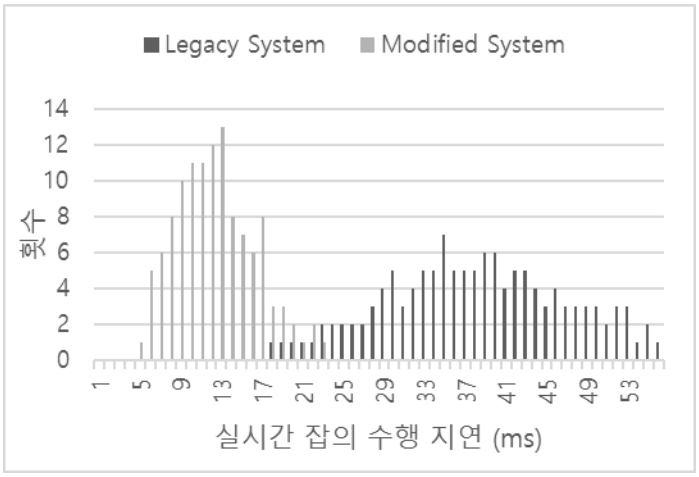

그림 4. 실험 결과

기법 적용 전후로 각각 100회씩 수행하였다.

그림 4는 실험결과를 나타낸다. 검은색 그래프는 기존 spark streaming 시스템의 결과값이며 회색 그래프는 제안 기법이 적용된 시스템의 결과값이다. 실험 결과 실시간 잡의 평균 수행 지연이 약 52.98% 감소함을 확인하였다.

5. 결론 및 향후 연구

본 논문에서는 자율주행 자동차를 지원하기 위한 실시간 클라우드에서 실시간 처리 요구사항을 충족하기 위한 선별적 자원 할당 기법을 제안하였다. 본 연구진은 Spark Streaming 2.1.0 시스템에 제안된 기법을 적용 후 실험을 통해 약 52.98%의 평균 수행 지연이 줄었음을 확인했다.

Acknowledgement

본 연구는 중소기업청의 기술혁신개발사업의 일환으로 수행하였음. [S2392603 , IoT 시계열 빅-데이터 실시간 저장 분석 및 시각화 플랫폼 기술 개발]

참 고 문 헌

[1] Sukhpal Singh and Inderveer Chana, "A survey on resource scheduling in cloud computing issues and challenges." *Journal of Grid Computing*, vol 14, pp.217-264, Feb. 2016

[2] Scott Schneider et al., "Dynamic Load Balancing for Ordered Data-Parallel Regions in Distributed Streaming Systems", *the 17th International Middleware Conference*, Nov. 2016

[3] Benjamin Hindman et al., "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center", *NSDI*, 2011

[4] Apache Spark Documentation. http://spark.apache.org/docs/latest/index.html

[5] Matei Zaharia et al., "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing", *NSDI*, 2012