한국컴퓨터정보학회 하계학술대회 논문집 제26권 제2호 (2018. 7)

장애 복구 응답성 향상을 위한 Spark Streaming 스케줄링 개선 메커니즘

황용하[O], 노순현[*]
[O*]서울대학교 전기정보공학부
e-mail: {yhhwang, shnoh}@redwood.snu.ac.kr[O*]

Improved Spark Streaming Scheduling Mechanism for Real-time Fault Recovery

Yongha Hwang[O], Soonhyun Noh[*]
[O*]Dept. of Electrical and Computer Engineering, Seoul National University

● 요 약 ●

최근 방대한 양의 스트림 데이터가 생산되면서 이를 실시간으로 처리하기 위한 프레임워크가 등장하였으며, 오픈 소스 영역에서 Spark Streaming이 주목받고 있다. Spark Streaming은 분산 환경에서 성능 향상을 위해 지연 스케줄링을 기반으로 응용을 수행하지만, 장애 발생 시 사용 가능한 태스크 슬롯을 빠르게 할당받지 못할 경우 장애 복구 시간이 지연될 수 있다는 문제점이 있다. 이러한 문제점을 해결하기 위해 본 논문에서는 실행자의 태스크 슬롯 보장을 통해 대기 시간 없이 즉시 할당할 수 있도록 하는 개선 메커니즘을 제안하였고, 실험 결과 장애 복구 응답성이 39.14% 개선됨을 확인하였다.

키워드: 스파크 스트리밍(Spark Streaming), 장애 복구(Fault recovery), 지연 스케줄링(Delay scheduling)

I. Introduction

최근 IoT 디바이스들이 다양한 산업 분야에서 사용되면서 방대한 양의 데이터를 연속적으로 생산하고 있다. 이에 따라 다양한 데이터 소스에서 연속적으로 생산되는 스트림 데이터를 실시간으로 분석하고 활용하려는 수요가 증가하였고, 이를 지원하기 위한 다양한 프레임워크가 등장하였다[1]. Spark Streaming은 스트림 데이터 처리를 위해서 가장 많이 사용되는 프레임워크 중 하나로써, 오픈 소스라는 장점을 이용하여 활발한 개발이 이뤄지고 있다.

Spark Streaming은 분산 환경에서 병렬적으로 데이터를 처리하기 때문에 빠른 데이터 처리가 가능하다는 장점이 있지만, 여러 노드를 병렬적으로 사용하기 때문에 단일 노드에 비해 장애가 발생할 확률이 높다는 문제가 있다. Spark Streaming에서 발생할 수 있는 장애는 드라이버 장애, 실행자 장애 2종류가 있다[2]. 드라이버 장애는 분산 환경에서 Spark Streaming 응용의 실행을 전반적으로 관리하는 드라이버 노드에 문제가 발생한 것이다. 반면 실행자 장애는 하드웨어 고장과 같은 이유로 실제 응용을 실행하는 실행자 노드에 문제가 발생하여, 해당 노드에서 실행 중이던 응용을 수행할 수 없게 된 것이다. 본 논문에서는 실행자 장애의 복구에 대해 다룬다.

Spark Streaming에서 장애 복구 응답성은 장애 인지 시점부터 장애 복구가 완료되는 시점까지의 종단 간 지연시간(end-to-end latency)으로 계산된다. 이 구간동안 드라이버에서 실행자의 장애를 인지하고, 해당 실행자에서 수행 중이던 태스크를 다른 실행자에 스케줄링하는 과정을 거친다. 이런 장애 복구 과정은 Spark Streaming 사용자들에게 보이지 않고 자동적으로 이루어진다.

불행하게도, 분산 환경에서는 다양한 디바이스들이 여러 네트워크에 연결되어 있고 각 디바이스를 동작시키는 시스템이 점점 복잡해지고 있기 때문에, 빠르게 장애를 복구하기 어려워지고 있다. 또한 장애는 비동기적으로 발생하기 때문에 Spark Streaming과 같이 별도의 장애 복구를 위한 스케줄러가 없는 경우, 장애 발생 시 사용 가능한 태스크 슬롯을 빠르게 할당받지 못한다면 장애 복구 과정에서 오랜 시간이 소요될 수 있다. 이는 장애 복구 응답성을 지연시킬 뿐 아니라, 전체 응용의 수행 시간을 지연시키는 문제를 일으킬 수 있다.

본 논문에서는 이러한 문제점을 해결하기 위해 장애 복구 응답성 향상을 위한 Spark Streaming 스케줄링 개선 기법을 제안한다. 개선 기법은 장애 복구를 수행할 실행자에 태스크 슬롯을 추가적으로 할당하여, 대기 없이 즉시 장애를 복구할 수 있도록 한다. 이는 장애 태스크의 대기시간을 없애 빠른 장애 복구를 가능하게 한다.

우리는 개선된 스케줄링의 장애 복구 성능 향상을 확인하기 위해 Spark Streaming이 설치된 분산 환경에서 실험을 진행하였다. 1개의 드라이버와 2개의 실행자 노드를 구성한 후, 1개 실행자 장애가

발생하는 경우에 대한 실험을 진행하였고, 장애 복구 시간이 39.14% 개선된 것을 확인하였다.

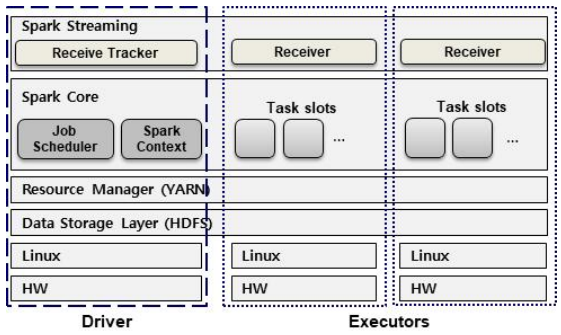
본 논문의 나머지 부분은 다음과 같이 구성된다. 2장에서는 Spark Streaming 프레임워크의 전체적인 구조와 런타임 동작에 관해 설명한다. 3장에서는 기존의 문제점과 해결 방안에 대해 간략히 설명한다. 4장에서는 구체적인 해결 방안을 설명하고, 5장에서 실험 결과를 보인 후 6장에서 결론을 맺는다.

II. Background

이 장에서 우리는 Spark Streaming 프레임워크의 전체적인 구조와 런타임 동작에 대해 알아본다.

2.1 Spark Streaming Architecture

Spark Streaming의 전체적인 구조는 그림 1과 같다. Spark Streaming은 Spark의 확장 컴포넌트로서, 스트림 데이터 처리를 위한 추가적인 기능을 사용해 동작한다.



Flg. 1. System architecture

분산 환경에서 동작하기 때문에 Spark와 동일하게 드라이버(Driver)와 실행자(Executor)로 구성되며, 드라이버에서는 Spark Streaming 응용과 분산 자원을 관리하기 위한 기능, 실행자에서는 스트림 데이터를 받아 처리하는 기능을 수행한다. Data storage layer는 분산된 노드가 같은 파일 시스템에 접근하여 데이터를 사용할 수 있도록 하고, resource manager는 각 분산 노드에서 사용할 자원을 분배하는 역할을 수행한다.

2.2 Runtime Behavior

2.2.1 Job 스케줄링

표 1에서 Spark Streaming의 런타임 동작의 이해를 돕기 위한 용어를 정의한다. Spark Streaming 응용은 여러 개의 job으로 구성되며, 드라이버는 job 스케줄러를 이용해 job의 실행 순서를 결정한다.

Spark Streaming의 런타임 동작 과정은 다음과 같다. (1) Job 생성, (2) DAG 생성, (3) Job 스케줄링, (4) 결과 반환. (1)에서는 Spark Streaming 응용 실행 시 RDD 생성부터 연산적용까지의 일련의 작업을 job 단위로 생성한다. (2)에서는 응용 프로그램 코드를 바탕으로 정점(Partition)과 간선(연산)을 생성한다. 생성된 DAG 내에서 셔플을 바탕으로 stage를 구분하고, stage 내에서 partition 연산을 나눠 태스크를 정의한다. (3)에서는 job 단위로 실행될 순서를 정의한다. 이 때, 해당 job을 위해 수행되어야 하는 태스크를 각 실행자에 할당한다. 마지막으로 (4)에서 드라이버는 실행자에서 수행된 결과를 반환 받고 실행을 종료한다.

Spark Streaming의 Job 스케줄링은 분산 환경에서 처리 성능을 높이기 위한 스케줄링 기법인 지연 스케줄링을 기반으로 수행된다[3]. 지연 스케줄링은 분산 환경에서 태스크 수행 시, 지역성을 고려하여 해당 태스크가 필요로 하는 데이터가 있는 노드에 태스크를 할당하는 기법이다. 실행 시간을 약간 늦추더라도 데이터가 있는 노드에 태스크를 할당함에 따라 분산 환경에서 데이터 교환에 따른 네트워크 트래픽을 줄여, 시스템의 전체적인 성능을 높여준다는 장점이 있다.

Table 1. Term definition

용어	정의
RDD	Resilient Distributed Dataset. 읽기 전용의 분산된 데이터들의 집합
Partition	분산 노드에서 RAM에 저장된 데이터 블록
DStream	단위 시간동안의 스트림 데이터를 나타내는 RDD 시퀀스
Job	RDD 생성부터 처리까지의 일련의 순서로, Spark Streaming 응용을 구성
Stage	동일한 셔플 의존성을 갖는 독립적인 태스크들의 집합. (셔플은 분산 환경에서 노드간 데이터 교환이 일어나는 연산)
Task	각 분산 노드에서 파티션을 처리하는 작업 단위
DAG	RDD를 정점으로, RDD 의존 관계를 간선으로 정의한 그래프

2.2.2 장애 복구 메커니즘

Spark Streaming은 장애 인지와 장애 복구 2단계로 나눠 장애를 처리한다. 장애 인지는 드라이버에서 주기적으로 실행자에 전송하는 heartbeat을 사용하며, 응답이 없을 시 장애로 인지한다. 장애가 발생하면 드라이버는 해당 실행자에서 수행되던 태스크를 정상 동작중인 다른 실행자로 옮겨서 수행시켜 장애를 복구 한다. Spark Streaming은 장애 처리를 위한 별도의 스케줄링이 없으며, 일반 태스크와 마찬가지로 장애 태스크도 지연 스케줄링을 기반으로 하여 실행하고자 하는 태스크를 해당 데이터가 있는 실행자에 배치하는 지역성을 고려하여 할당한다.

III. Problem Definition

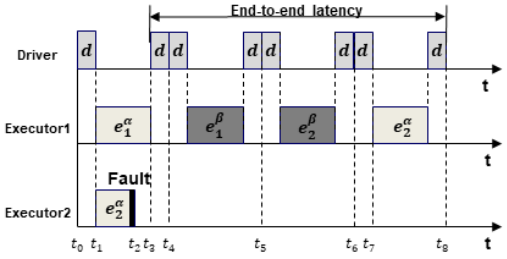
이 장에서는 장애 인지부터 장애 복구까지의 종단 간 지연시간에 대해 설명하고, 예제를 통해 장애 발생 시 Spark Streaming 스케줄링 문제를 밝힌다.

Spark Streaming 응용 S는 여러 개의 job으로 구성되어 있고, 하나의 job은 여러 개의 태스크로 구성된다. 드라이버에서 수행되는 태스크를 d, 실행자에서 수행되는 태스크를 e_n이라 할 때, 응용 $S = \{d, e_1, e_2, \dots e_n\}$로 표현할 수 있다. 분산 환경에서 각

태스크는 지연 스케줄링에 따라 지역성 있는 실행자에서 수행된다. 하지만 장애 발생 시, 분산 환경에서 Spark Streaming의 성능 향상을 위해 적용된 지연 스케줄링이 오히려 성능 저하를 일으키는 문제가 될 수 있다.

그림 2는 구체적인 문제 상황을 설명한다. 그림 2와 같이 1개의 드라이버와 2개의 실행자가 있고, 각 job은 2개의 태스크로 구성되며 각 실행자는 1개의 태스크 슬롯을 가지고 있고, t_2 시점에 실행자 2에 장애가 발생한 상황이라고 하자.

$[t_0, t_1)$에서 드라이버는 실행자 1, 2에 J_a의 태스크 e_1^a, e_2^a를 각각 할당한다. 각 실행자는 태스크 수행 결과를 드라이버에 반환하는데, t_2 시점에 실행자 2에 장애가 발생하면, e_2^a는 처음부터 다시 수행되어야 한다. $[t_4, t_6)$에 J_β의 수행이 끝나면 $[t_6, t_8)$에 e_2^a를 다시 수행한다.



Flg. 2. Gantt chart of fault recovery

여기서 장애 인지 시점 t_3부터 복구 완료 시점 t_8까지의 시간 차이를 태스크 e_2^a 장애 복구 종단 간 지연시간이라 한다. 이 때, 문제는 실행자 1에 e_2^a에 대한 지역성이 없을 때이다. 즉, e_2^a 수행을 위한 데이터가 실행자 1에 없을 경우, 다른 실행자에서 데이터를 복사해오거나, 데이터를 가진 다른 실행자의 태스크 슬롯이 생길 때까지 대기한다[4]. 따라서 지역성이 없다면, 그림 2처럼 t_6 시점에 e_2^a를 바로 실행할 수 없고, Spark Streaming 스케줄링 메커니즘에 따라 지역성 있는 실행자의 태스크 슬롯이 사용 가능할 때 까지 대기해야한다. 이는 해당 실행자에서 수행 중이던 다른 태스크의 수행을 지연시킬 수 있고, 전체적인 분산 시스템의 성능 저하를 일으키는 문제가 된다.

따라서 본 논문에서는 장애 발생 시 지역성 있는 태스크 슬롯에서의 동작을 보장하여 스케줄링으로 인한 지연을 최소화함으로서, 장애 복구의 종단 간 지연시간을 줄이는 것을 목표로 한다.

IV. Advanced Fault Recovery Mechanism

이 장에서는 장애 복구 응답성 향상을 위해 제안한 기법에 대해 개관한 후 세부 내용을 설명한다.

본 논문의 핵심 아이디어는, 장애 발생 시 지역성이 있는 실행자의 태스크 슬롯 보장을 통하여 장애 태스크 대기 시간을 최소화 하는 것이다. 이를 통하여 장애 복구의 종단 간 지연시간을 줄일 수 있다.

그림 3은 Spark Streaming 기반 시스템의 구성과 그 동작 과정을 나타낸다. 장애 발생시 Spark Streaming은 드라이버 내부의 컴포넌트를 복합적으로 사용하여 장애를 복구한다. 본 논문에서는 지역성 있는 실행자에 태스크 슬롯을 할당을 보장하기 위해 Spark Streaming에서 제공하는 기능을 이용하여 컴포넌트의 동작을 수정한다. 본 연구에서 제안하는 기법 관련 동작을 수행하는 컴포넌트는 block manager, job scheduler, cluster manager 3가지이며, 각각 장애 발생 인지, 지역성 관리, 태스크 슬롯 할당 역할을 수행한다.

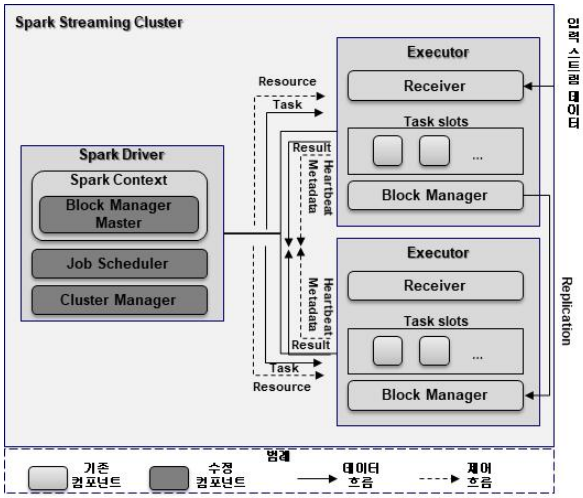


Fig. 3. Spark Streaming 기반 시스템 동작 개관

드라이버의 spark context는 실행자 동작을 관리하는 역할을 수행하며, 그 중에서도 block manager master는 실행자 장애 여부를 확인하는 역할을 수행한다. 그 기능은 spark.executor.heartbeat Interval 값을 통해 설정할 수 있으며, 값이 들어오지 않을 경우 장애 발생 시점으로 판단한다. 장애가 발생하면 job scheduler는 내부적으로 장애가 발생한 실행자에서 수행 중이던 태스크를 파악하고, 해당 태스크에 지역성이 있는 실행자를 찾아 태스크를 배치하게 된다. 이 때, job scheduler는 태스크에 대한 지역성 목록을 유지하고 있기 때문에 태스크를 수행할 실행자를 확인한다. 마지막으로 클러스터 매니저는 실행자에 CPU, 메모리 등의 자원을 할당해주는 역할을 하며, 이 컴포넌트의 SPARK_WORKER_CORES 옵션을 통해 각 실행자에서 장애 태스크가 수행될 수 있도록 태스크 슬롯 개수를 조절한다.

V. Experimental Evaluation

이 장에서는 실험과 그 결과를 통해 제안된 기법의 효과를 검증한다.

5.1 Experimental Setup

본 연구진은 제안된 해결 방안을 Spark Streaming에 구현하고 3대의 컴퓨팅 노드로 구성된 분산 환경에서 실험을 통해 장애 복구 응답성 개선 여부를 검증하였다. 구체적인 실험 환경은 표 2와 같다.

본 실험을 통해 검증하고자 하는 가설은 제안된 기법을 통해 장애 복구 종단 간 지연시간이 줄어드는지 여부이다. 대조군은 제안된 방법이 적용되지 않은 시스템이며 실험군은 제안된 방법이 적용된 시스템이다. 조작 변인은 태스크 슬롯 개수로, Spark Streaming 응용 수행 중 장애가 발생하였을 때, 지역성 있는 실행자에 태스크 슬롯 개수에 따른 수행시간의 차이를 비교하였다.

Table 2. 실험 환경

환경		드라이버	실행자
하드웨어	CPU	Intel i7-4770	Intel i7-3770
	메모리	32GB	6GB
	스토리지	SSD 500GB	HDD 1TB
소프트웨어	데이터 분석 프레임워크	Spark Streaming 2.2.0	
	분산 파일 시스템	Hadoop 2.7.1	
	운영체제	Linux 16.04.1	

5.2 Experimental Result

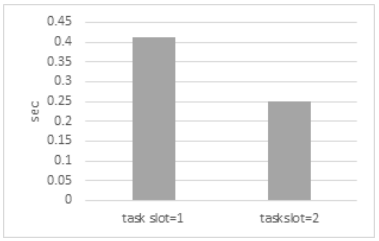


Fig. 4. Experimental result

실험은 대조군과 실험군에 대해 조작변인을 변경해가며 진행하였고, wireshark를 사용하여 장애 인지부터 복구까지의 종단 간 지연시간 을 10회 측정한 후 평균을 계산하였다. 그 결과 제안된 기법을 적용할 경우 그림 4와 같이 장애 복구 시간이 평균 0.25s로 기존 0.41s보다 약 39.14% 더 빠르게 장애를 복구하는 것을 확인하였다.

VI. Conclusion

본 논문에서는 Spark Streaming 프레임워크에서 장애 복구 응답성 개선을 위한 스케줄링 메커니즘을 제안하였다. 기존 Spark Streaming의 스케줄링은 분산 환경에서 처리 성능 향상을 위해 지연 스케줄링을 사용하였지만, 이는 장애 태스크 복구 시, 지역성 있는 태스크 슬롯을 할당받지 못하면 실행이 지연되어 전체 시스템의 성능 하락을 발생시키는 문제가 있다. 따라서 본 논문에서는 이를 개선한 기법을 Spark Streaming 2.2.0 기반 시스템에 적용하였고, 실험을 통해 장애 복구 시간이 39.14% 개선된 것을 확인하였다.

ACKNOWLEDGMENT

"본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 대학 ICT연구센터육성지원사업의 연구결과로 수행되었음" (IITP-2018-2013-1-00717)

REFERENCES

[1] http://www.complexevents.com/2016/06/15/proliferation-of-open-source-technology-for-event-processing/
[2] https://spark.apache.org/docs/latest/streaming-programming-guide.html
[3] Zaharia, Matei, et al. "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling." Proceedings of the 5th European conference on Computer systems. ACM, 2010.
[4] https://spark.apache.org/docs/latest/job-scheduling.html.
[5] Jeongho Kim, et al. "Fault Recovery Time Estimation and Checkpointing Technique for Real-Time Fault Recovery in Spark Streaming System", KCC, 2016.