KIPS Trans. Softw. and Data Eng.
Vol.7, No.8 pp.297~306 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2018.7.8.297

Reconfiguration of Apache Storm for InfiniBand Communications

Seokwoo Yang† · Siwoon Son†† · Yang-Sae Moon†††

ABSTRACT

In this paper, we address how to apply Apache Storm, a distributed stream processing framework, to InfiniBand, a high performance communication device. An easy way to run Storm on InfiniBand is to simply use IPoIP (IP over InfiniBand). However, this method causes a serious CPU load on the node, which is caused by frequent context switches and buffer copies. To solve this problem, we propose a new communication method using InfiniBand's Remote Direct Memory Access (RDMA) function in Storm. First, we design and implement RJ-Netty (RDMA/JXIO Netty), a new framework that replaces Netty, the legacy framework, to exploit RDMA functionality. Second, we reimplement the related classes so that Storm can use both existing Netty and new RJ-Netty. Third, we extend the JXIO server functionality so as to support multi-threading to maximize the performance of RJ-Netty. Experimental results show that the proposed RJ-Netty significantly reduces CPU load while improving message throughput compared to IPoIB as well as Ethernet. This paper is the first attempt to run Apache Storm on InfiniBand, and we believe that it is an excellent research result that improves the performance of Storm by using InfiniBand RDMA.

Keywords : Apache Storm, InfiniBand, IPoIB, RDMA, Netty, RJ-Netty

InfiniBand RDMA 통신을 위한 Apache Storm의 재구성

양 석 우† · 손 시 운†† · 문 양 세†††

요 약

본 논문에서는 분산 스트림 처리 프레임워크인 Apache Storm을 고성능 통신 장비인 InfiniBand에 적용하는 방안을 다룬다. InfiniBand 상에서 Storm을 동작시키는 쉬운 방법은 IPoIB (IP over InfiniBand)를 사용하는 것이다. 그러나 이 방법은 노드에 심각한 CPU 부하를 발생시키는데, 이는 잦은 문맥 전환과 버퍼 복사에서 기인하는 것으로 나타났다. 이를 해결하기 위해, Storm에서 InfiniBand의 RDMA (Remote Direct Memory Access) 기능을 사용하는 새로운 통신 방식을 제안한다. 첫째, Storm에서 RDMA 기능을 이용하기 위해, 기존 통신 프레임워크인 Netty를 대체하는 새로운 프레임워크인 RJ-Netty (RDMA/JXIO Netty)를 설계 및 구현한다. 둘째, Storm이 기존 Netty와 RJ-Netty를 모두 사용할 수 있도록 관련 클래스들을 개선한다. 셋째, RJ-Netty의 성능을 최대화하기 위해 멀티스레드를 지원하도록 JXIO 서버 기능을 개선한다. 실험 결과, 제안한 RJ-Netty는 Ethernet은 물론 IPoIB에 비해서 메시지 처리량을 향상시키면서도 CPU 부하를 크게 줄인 것으로 나타났다. 본 논문은 Apache Storm을 InfiniBand 상에서 동작시킨 최초의 시도로, 고성능의 InfiniBand RDMA를 사용하여 Storm의 처리 성능을 향상시킨 우수한 연구 결과라 사료된다.

키워드 : Apache Storm, InfiniBand, IPoIB, RDMA, Netty, RJ-Netty

1. 서 론

최근 데이터의 양과 생성 속도가 증가하면서 Apache Hadoop[1]과 같은 배치 처리 프레임워크보다, Apache Storm[2-4], Apache Spark[5], Apache S4[6], Apache Flink[7]와 같이 대용량 데이터 스트림을 실시간 처리할 수 있는 프레임워크들이 주목받고 있다. 이중, Storm은 대용량 데이터 스트림을 빠르게 처리할 수 있는 실시간 분산 처리 프레임워크이다. 또한, Storm과 같은 분산 처리 프레임워크에서 노드 간 통신 속도를 최대화하기 위해 InfiniBand[8-13]를 도입하는 사례가 증가하고 있다. InfiniBand는 고성능 컴퓨팅(High-Performance Computing: HPC) 클러스터의 노드를 상호 연결하기 위해 설계된 업계 표준 스위치 패브릭으로, 높은 대역폭(bandwidth)과 낮은 전송 지연 시간을 보장하는 범용 통신 장비이다. 본 논문에서는 고성능 네

※ 이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원(No. 2016-0-00179, 데이터 스트림 정제를 위한 지능형 샘플링 및 필터링 기술 개발)과 2018년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2017R1A2B4008991).
† 준 회 원 : 강원대학교 컴퓨터과학과 석사
†† 준 회 원 : 강원대학교 컴퓨터과학과 박사과정
††† 종신회원 : 강원대학교 컴퓨터과학과 교수
Manuscript Received : March 5, 2018
Accepted : March 27, 2018
* Corresponding Author : Yang-Sae Moon(ysmoon@kangwon.ac.kr)

※ This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/ licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

트워크 장비인 InfiniBand를 사용하여 Storm의 처리 성능을 크게 향상시키고자 한다. 이를 구현할 수 있는 간단한 방법은 InfiniBand에서 제공하는 IPoIB (IP over InfiniBand)를 사용하는 것이다. IPoIB는 Ethernet 장비를 사용하는 것과 동일하게 IP 주소를 통해 쉽게 InfiniBand 장비를 사용할 수 있도록 지원하는 기능이다.

그러나, IPoIB 기반 Storm은 **CPU 과부하 문제**로 인해 InfiniBand의 성능을 제대로 활용하지 못하게 된다. Storm은 분산 서버에서 다수의 워커(worker) 프로세스를 통해 대용량 데이터 스트림을 처리하는 구조를 가지며, 이러한 워커 간 통신은 TCP/IP 프로토콜 기반의 Netty[14]로 구현되어 있다. 이 같은 Storm 네트워크 구조의 분석 결과, IPoIB 기반 Storm에서 발생하는 CPU 과부하 문제의 주요 원인은 다음 두 가지로 밝혀졌다.

■ **문맥 전환**[15]: Netty는 비동기 이벤트 기반 I/O 프레임워크로 워커에서 특정 스레드를 통해 메시지를 전송하면 제어가 운영체제로 넘어가고, 스레드는 전송이 완료될 때까지 대기한다. 이후, 전송이 완료되면 해당 스레드는 다시 활성화되어 메시지 전송 완료에 관한 메타데이터를 기록한다. 이러한, 스레드의 대기 및 활성화 과정의 문맥 전환은 많은 CPU 자원을 요구한다.

■ **버퍼 복사**: TCP/IP 프로토콜 기반의 메시지 송수신을 위해서는 프로세스와 운영체제 사이의 버퍼 복사, 운영체제 내에서 버퍼 복사 작업이 필요하다. 이러한 두 가지 버퍼 복사 작업은 많은 CPU 자원을 요구한다.

결국, 다수의 워커가 통신하는 Storm은 잦은 문맥 전환과 버퍼 복사로 인해 CPU 과부하 문제를 야기한다. 특히, 이는 Storm의 병렬성이 증가할수록 심해지는데, 그 이유는 워커와 워커 사이에 더 많은 통신이 발생하기 때문이다. 이와 같은 CPU 과부하 문제는 InfiniBand의 성능을 제대로 발휘하지 못하는 원인이 될 뿐 아니라 전체 시스템의 성능을 크게 저하시키는 원인이 된다.

CPU 과부하 문제를 해결하기 위해, 본 논문에서는 InfiniBand의 RDMA(Remote Direct Memory Access) 기능을 이용하여 Storm을 재구성한다. 구체적으로, 종단 노드 간 데이터 송수신 시, 운영체제를 거치지 않고 호스트 메모리에서 원격지 메모리로 직접 데이터를 전송할 수 있는 InfiniBand의 RDMA 기능을 사용한다. 제안 방법의 주요 설계 내용은 다음과 같다. 첫째, JXIO[16]를 사용하여, RDMA 기능을 사용하면서도 기존 Netty와 동일한 기능을 수행하는 새로운 네트워크 통신 계층인 RJ-Netty (RDMA/JXIO Netty)[1)]를 구축한다. 여기서, JXIO는 InfiniBand에서 RDMA 기능을 구현할 수 있도록 제공하는 Verbs 인터페이스에 대한 자바 API이다. 둘째, Storm을 실행할 때 기존 Netty와 RJ-Netty를 선택할 수 있도록 하여 TCP/IP와 RDMA 통신 모드를 모두 사용할 수 있도록 Storm의 소스코드를 개선한다. 셋째, 제안한 방법의 성능을 최적화하기 위해 멀티스레드 처리를 위한 JXIO 서버 구조를 개선한다. 본 논문에서는 이들 세 가지 설계 내용을 실제 Storm에 적용하여 구현하고, Storm이 실제로 RDMA 상에서 정확히 동작함을 실험으로 확인한다.

Ethernet, IPoIB, RDMA 기반 Storm의 비교 실험 결과, 제안한 방법은 Storm의 메시지 처리 성능을 향상시키면서도 CPU 과부하 문제를 해결한 것으로 나타났다. RJ-Netty를 사용한 RDMA 기반 Storm은 처리량 측면에서 IPoIB에 비해 최대 1.25배까지, Ethernet에 비해 최대 10.2배까지 성능을 향상시켰다. 특히, RDMA는 IPoIB에 비해 메시지 처리량을 개선하였음에도 불구하고, CPU 부하는 IPoIB 대비 최대 2.6배까지 줄인 것으로 나타났다.

본 논문의 구성은 다음과 같다. 먼저, 제2절에서는 본 연구의 배경이 되는 Storm, InfiniBand, JXIO를 차례로 설명한다. 제3절에서는 Storm의 네트워크 구조를 분석하고, 이를 통해 IPoIB 기반 Storm에서 발생하는 CPU 과부하 문제의 원인을 도출한다. 제4절에서는 RDMA 기반 Storm의 RJ-Netty 구조를 설계하고 구현한다. 또한, JXIO 서버 구조를 멀티스레드 환경에 적합하도록 RJ-Netty를 최적화한다. 제5절에서는 제안하는 방법이 CPU 과부하 문제를 해결하고 메시지 처리 성능을 향상시킬 수 있음을 보인다. 마지막으로, 제6절에서 본 논문을 요약하고 결론을 맺는다.

2. 관련 연구

2.1 Apache Storm

Apache Storm은 대용량 데이터 스트림을 안정적으로 처리할 수 있는 실시간 분산 처리 프레임워크이다. Storm은 크게 물리적 구조와 논리적 구조로 구분할 수 있다. 먼저, Storm 클러스터의 물리적 구조는 하나의 마스터 노드와 다수의 슬레이브 노드를 갖는 마스터/슬레이브 구조이다. 즉, Storm의 님버스(Nimbus)는 마스터 노드, 슈퍼바이저(Supervisor)는 슬레이브 노드에 해당한다. 또한 님버스와 슈퍼바이저 간의 상태 정보를 공유하기 위해 서드파티(third-party) 개념으로 Apache Zookeeper[17, 18]를 사용한다.

Fig. 1은 Storm 클러스터의 물리적 구조를 보여준다. 그림 1에서 님버스는 쓰리프트(Thrift)[19]를 이용해 슈퍼바이저에게 토폴로지를 배포하고, 주기적으로 전송되는 하트비트(heartbeat)를 통해 모든 슈퍼바이저의 상태를 관리한다. 이때, 님버스가 특정 슈퍼바이저의 하트비트를 받지 못하거나 서비스 불능 상태라 판단하면, 해당 슈퍼바이저가 처리 중이던 작업을 회수하여 클러스터 내의 다른 정상 슈퍼바이저에게 재할당한다. 그리고, 슈퍼바이저는 님버스로부터 할당받은 작업을 수행하며, 주기적으로 하트비트를 통해 님버스에게 상태를 보고한다. 이러한 님버스와 슈퍼바이저의 통신은 슈퍼바이저가 메타데이터를 Zookeeper에 기록하고, 님버스가 이를 읽음으로써 이루어진다.

---

1) 본 논문에서 설계 및 구현한 RJ-Netty의 모든 소스 코드는 GitHub에 오픈 소스로 공개되어 있다. GitHub 주소는 https://github.com/dke-knu/i2am/tree/master/rdma-based-storm이다.

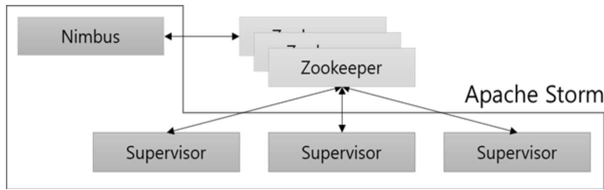


Fig. 1. Physical Structure of an Apache Storm Cluster

다음으로, Storm은 데이터 스트림을 처리하기 위해 Fig. 2와 같은 논리적 구조를 사용하며, 이들 컴포넌트에 대한 설명은 다음과 같다.

■ 토폴로지(Topology): Storm에서 작업할 내용의 분산 처리 구조를 나타낸다. 데이터의 입력부터 출력까지 일련의 작업을 정의하며, 여러 개의 스파우트와 볼트로 구성된다.

■ 스파우트(Spout): 데이터 스트림 소스와 연결을 맺고, 입력 튜플을 Storm의 기본 데이터 구조인 튜플(tuple)로 변환하여 다음 볼트로 전송한다.

■ 볼트(Bolt): 스파우트 혹은 다른 볼트로부터 수신한 데이터를 정의된 로직에 따라 처리하여 다음 볼트로 전송하거나 출력 또는 저장한다.

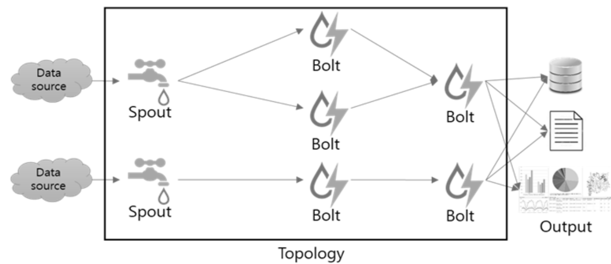


Fig. 2. Logical Structure of Apache Storm

Storm의 주요 특징 중 하나는 병렬성이다. 즉, 같은 작업을 수행하는 스파우트 또는 볼트를 서로 다른 서버에서 병렬로 동작시킬 수 있다. Storm의 병렬성은 서버를 의미하는 노드, 프로세스를 의미하는 워커(Worker), 스레드를 의미하는 익스큐터(Executor), 스파우트 또는 볼트의 객체를 의미하는 태스크(Task)가 있다. 따라서, 노드는 여러 개의 워커를 가질 수 있고, 워커는 여러 개의 익스큐터를 가질 수 있으며, 익스큐터는 여러 개의 태스크를 가질 수 있는 계층적 구조이다.

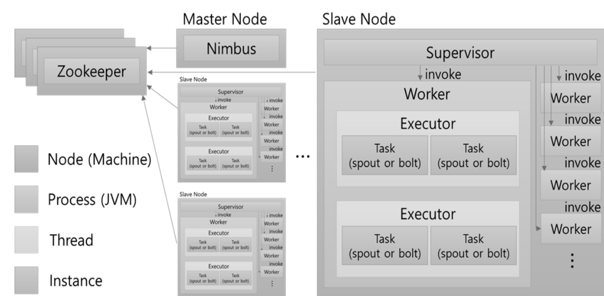


Fig. 3. Detailed Structure of an Apache Storm Cluster

여기서, 워커는 실제로 토폴로지를 처리하는 자바 프로세스로 익스큐터 외의 몇몇 스레드를 실행하여 튜플의 처리 및 송수신을 수행한다. 그리고, 익스큐터는 워커에서 동작하는 자바 스레드로 스파우트 또는 볼트의 객체를 생성하여 튜플의 처리를 수행한다. Fig. 3은 이와 같은 Storm 클러스터의 전체적인 세부 구조를 보여준다.

2.2 InfiniBand

InfiniBand는 높은 대역폭과 낮은 전송 지연시간을 보장하는 고성능 통신 장비이다. InfiniBand의 주요 기능 중 하나는 RDMA로, 데이터 송수신시 운영체제를 거치지 않고 호스트 메모리에서 원격지 메모리로 직접 데이터를 전송한다. 이러한 RDMA를 사용하기 위해 InfiniBand는 Verbs라 불리는 인터페이스를 제공한다. Verbs는 저수준 통신 인터페이스로써 상위 계층의 애플리케이션은 이를 통해 RDMA 통신을 구현할 수 있다. 이와 같이 RDMA 통신을 구현함으로써, zero-copy, kernel bypass, no CPU involvement의 장점을 발휘할 수 있다. 이에 따라, 버퍼 복사, 문맥 전환과 같은 중간 과정의 오버헤드가 크게 줄어들어, 데이터 처리량 증가와 전송 지연시간 감소의 효과를 얻을 수 있다.

다음으로, InfiniBand는 IPoIB 계층을 제공함으로써, IP 주소를 통해 InfiniBand 장비에 접근할 수 있는 네트워크 인터페이스를 제공한다. 일반적으로, Ethernet 장비를 eth0, eth1로 표현하는 것처럼, InfiniBand 장비는 ib0, ib1과 같이 표현한다. 그리고, Ethernet과 마찬가지로 비신뢰형 데이터그램(unreliable datagram) 모드와 신뢰형 연결(reliable connection) 모드를 지원한다. IPoIB는 IP 계층 이용으로 InfiniBand 사용이 편리한 장점이 있는 반면에, 네트워크 소프트웨어 스택을 사용하기 때문에 RDMA에 비해서는 성능이 다소 떨어지는 단점이 있다.

2.3 JXIO

JXIO는 하드웨어 가속을 위해 비동기식 메시지 전송 라이브러리인 Accelio[20, 21]의 자바 인터페이스이다. Accelio는 RDMA 뿐만 아니라 TCP/IP, 공유 메모리와 같이 다양한 통신 기술을 효율적이고 쉽게 구현할 수 있는 API를 제공한다. 또한, Accelio는 CPU 경합과 잠금을 최소화하고, 데이터의

Table 1. Accelio Components for RDMA Communications

Type	Classes/Interfaces
Class	■ EventQueueHandler ■ Msg ■ MsgPool ■ ClientSession ■ ServerSession ■ ServerPortal
Interface	■ EventQueueHandler.Callbacks ■ ClientSession.Callbacks ■ ServerSession.Callbacks ■ ServerPortal.Callbacks

zero-copy를 가능하게 함으로써 메시지 전송 및 CPU 병렬 처리를 극대화할 수 있도록 설계되었다. Table 1은 RDMA 통신을 위한 Accelio의 기술 요소들과 이에 대한 설명으로, Accelio의 자바 인터페이스인 JXIO 또한 동일한 요소를 갖는다.

3. IPoIB 기반 Storm 설계와 문제점 분석

3.1 Storm의 메시지 처리 구조 분석

사용자가 Storm 클러스터에 토폴로지를 제출한 이후의 처리 과정은 다음과 같다. 먼저, 토폴로지를 받은 님버스는 쓰리프트를 이용해 슈퍼바이저에게 토폴로지를 배포한다. 이어서, 슈퍼바이저는 토폴로지의 설정에 명시된 개수만큼의 워커를 생성하고, 생성된 워커는 서로 통신을 하며 메시지를 처리한다. 이때, 워커는 내부적으로 크게 세 개의 스레드로 구성된다. 본 논문에서는 각 스레드를 워커 수신 스레드(Worker Rx Thread), 익스큐터(Executor) 스레드, 워커 송신 스레드(Worker Tx Thread)로 표현한다.

Fig. 4는 워커 프로세스의 구조와 Storm에서 메시지를 처리할 때의 작업 흐름을 나타낸다. 그림에서 점선은 한 워커 내에서의 여러 스레드 간 통신으로, LMAX Distruptor[22]를 사용하여 구현되어 있다. 본 논문에서는 이를 워커 내 통신(intra-worker communication)이라 부른다. 다음으로, 실선은 Storm 클러스터에서 동작하는 여러 워커 간 통신으로 Netty를 사용하여 구현되어 있다. 본 논문에서는 이를 워커 간 통신(inter-worker communication)이라 부른다. Storm은 토폴로지 처리를 종료할 때까지 Fig. 4의 ■-■ 과정을 반복하며 데이터 스트림을 처리한다. 각 과정에 대한 자세한 설명은 다음과 같다.

① **워커 프로세스 생성 및 포트 할당**: 사용자가 Storm 클러스터로 토폴로지를 제출하면 님버스는 쓰리프트를 이용해 토폴로지를 슈퍼바이저에게 배포한다. 그리고, 슈퍼바이저는 토폴로지를 읽은 후 필요한 수만큼 워커 프로세스를 생성하고 워커마다 서로 다른 포트를 할당한다. 그림 4에서는 Storm의 기본 환경설정으로 6700부터 포트가 할당되어 있다.

② **워커 수신 스레드**: 생성된 워커는 먼저 워커 수신 스레드를 생성한다. 워커 수신 스레드는 워커가 할당 받은 포트를 통해 연결 요청 이벤트를 기다린다. 이후 모든 연결이 완료되면, 워커 수신 스레드는 네트워크를 통해 전달된 메시지를 워커의 수신 큐(RX QUEUE)로 받은 후, 익스큐터의 입력 큐에 저장한다.

③ **워커 송신 스레드**: 이어서 워커는 다수의 워커 송신 스레드를 생성한다. Storm 클러스터에서 n개의 워커가 동작 중일 때, 각 워커는 $n-1$개의 워커 송신 스레드를 생성하여 워커 송신 스레드와 외부 워커의 관계를 1:1로 설정한다. 워커는 주기적으로 워커 송신 스레드에게 튜플의 송신을 요청하고, 요청을 받은 워커 송신 스레드는 연결 상태에 있는 워커의 워커 수신 스레드로 튜플을 전달한다.

④ **사용자 로직 스레드**: 모든 워커들의 연결이 완료되면, 워커는 익스큐터를 생성한다. 익스큐터는 먼저 사용자 로직 스레드를 생성하고, 이를 통해 스파우트 혹은 볼트 태스크에 정의된 내용을 다음과 같이 처리한다.

- **스파우트**: 익스큐터에서 실행되는 태스크가 스파우트이면, 사용자 로직 스레드는 스파우트에 정의된 내용에 따라 데이터 소스와 연결을 맺고, 데이터를 튜플로 변환하여 익스큐터의 출력 큐(OUT QUEUE)에 저장한다.
- **볼트**: 익스큐터에서 실행되는 태스크가 볼트이면, 사용자 로직 스레드는 입력 큐에서 튜플을 가져와 볼트에 정의된 내용에 따라 튜플을 처리하고, 그 결과 새로 생성된 튜플을 출력 큐에 저장한다.

⑤ **익스큐터 전송 스레드**: 익스큐터는 사용자 로직 스레드와 함께 자신의 전송 스레드를 생성한다. 익스큐터 전송 스레드는 출력 큐에서 튜플을 가져와 다음 볼트가 동작 중인 워커로 전송한다. 이때, 다음 볼트가 같은 워커의 다른 익스큐터에서 동작 중이면, LMAX Disruptor를 통해 해당 튜플을 전송한다. 만약 다음 볼트가 다른 워커에서 동작 중이면, 네트워크 전송을 위해 워커 전송 스레드의 전송 큐에 튜플을 보내며, 이때 Netty를 사용한다.

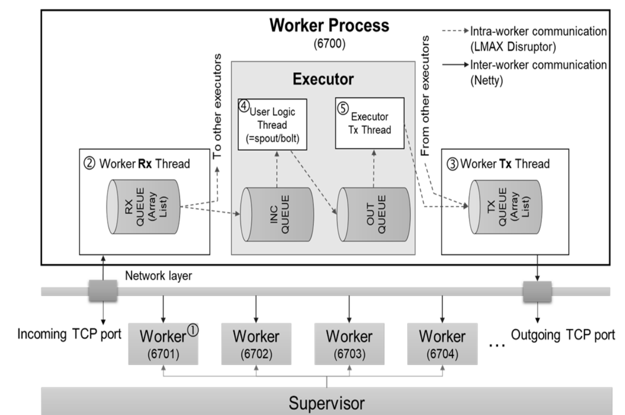


Fig. 4. Structure of a Worker Process in Storm

3.2 IPoIB 기반 Storm의 CPU 과부하 문제

본 절에서는 기존 Netty 통신의 CPU 과부하 문제를 설명한다. Netty는 비동기 이벤트 기반 I/O 프레임워크로, 워커 전송 스레드가 메시지를 다른 워커로 전송하면 제어는 운영체제로 넘어가고, 해당 워커 전송 스레드는 대기 상태가 된다. 이후, 메시지 전송이 완료되면, 워커 전송 스레드는 다시 활성화되어, 관련 메타데이터를 저장하고 다음 메시지 전송 요청이 있을 때까지 다시 대기한다. 이 과정에서 발생하는 스레드의 **문맥 전환**은 CPU 자원을 필요로 한다. 또한, 그림 5와 같이 TCP/IP 프로토콜을 기반으로 메시지를 전송할 경우, **운영체제로의 버퍼 복사**와 **운영체제 내에서의 버퍼 복사**가 다수 발생한다. 이 또한 모두 CPU 자원을 필요로 한다. 즉, 실시간 스트림 처리를 위한 Storm은 수많은 네트워크 통신이 이루어지고, 이로 인한 잦은 문맥 전환과 버퍼 복사는 심각한 CPU 과부하 문제를 야기한다. 특히, Storm의 병렬성이 증가할수록

CPU 과부하 문제는 더욱 심각해진다. 또한, CPU 과부하로 인해 워커 내의 통신 모듈은 CPU 자원이 할당될 때까지 기다려야 하고, 결과적으로 처리 지연시간이 증가하고 메시지 처리량이 감소한다.

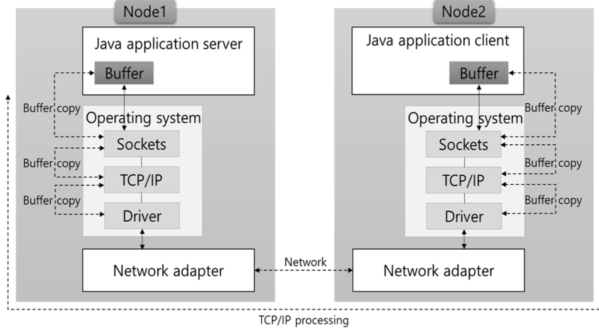


Fig. 5. Procedure of Buffer Copying in TCP/IP Protocol

CPU 과부하 문제를 해결하기 위해 본 논문에서는 IPoIB 대신 InfiniBand에서 제공하는 RDMA 기술을 Storm에 적용하고자 한다. Fig. 6은 TCP/IP 통신과 RDMA 통신의 차이점을 나타낸다. 그림에서 볼 수 있듯이, TCP/IP 통신과 다르게 RDMA 통신은 네트워크 어댑터가 원격 애플리케이션의 메모리에 직접 접근하여 데이터를 조작한다. 이에 따라, 문맥 전환 및 버퍼 복사와 같이 CPU 자원을 필요로 하는 중간 과정을 생략함으로써 CPU 부하를 줄이고 데이터 통신의 성능을 개선할 수 있다. 따라서, 본 논문에서는 CPU 과부하 문제 해결을 위해 InfiniBand RDMA 기반 Storm의 네트워크 구조를 설계하고 구현한다.

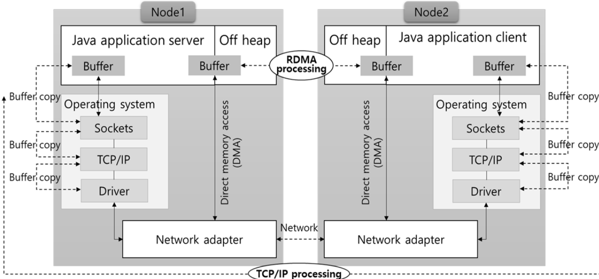


Fig. 6. Comparison of TCP/IP and RDMA Processing Procedures

4. InfiniBand RDMA 기반 Storm 네트워크 구조

본 절에서는 기존 Storm에서의 Netty 기반 TCP/IP 통신이 아닌, JXIO 기반의 RDMA 네트워크 통신 계층인 RJ-Netty를 새롭게 설계 및 구현한다. 이를 위해, 기존 Storm에서 Netty로 구현된 메시지 처리 구조의 소스 코드를 분석하고, 이를 바탕으로 RDMA 기반 Storm의 워커 간 통신 구조인 RJ-Netty를 제시한다.

4.1 RDMA 기반 Storm 메시지 처리 구조

본 절에서는 제안하는 RJ-Netty의 메시지 처리 구조를 설명한다. RJ-Netty는 Fig. 7과 같이 기존 Storm 클래스의 개선과 JXIO를 사용한 새로운 메시지 전송 계층의 구현을 포함한다. 그림을 보면, RJ-Netty 통신을 위해 `TransportFactory` 클래스를 변경하고, Netty의 기능을 대신할 클래스들을 JXIO로 새롭게 설계하였음을 알 수 있다. 이 같은 변경에 대한 자세한 설명은 다음과 같다.

■ `TransportFactory` 클래스: 이 클래스는 메시지 플러그인의 `Context` 객체를 워커에 반환하는 역할을 한다. 기존 Storm에서 메시지 플러그인을 설정할 때, `org.apache.storm.messaging.netty.Context`와 같이 `Context` 클래스의 패키지 경로 전체를 기입해야 한다. 이는 Storm의 환경 설정에 많은 불편함이 따르므로, 새로 구현한 RJ-Netty의 `Context` 객체와 Netty의 `Context` 객체를 “JXIO” 혹은 “Netty” 키워드로 간단히 선택할 수 있도록 새로운 환경 변수를 제공한다.

■ `Context` 클래스: Storm에서 메시지 플러그인을 새롭게 구현하기 위해서는 `IContext` 인터페이스를 구현한 `Context` 클래스가 필요하다. 따라서, RDMA 통신을 위해 JXIO로 구현된 `Server` 객체 및 `Client` 객체를 워커가 얻을 수 있도록 RJ-Netty를 위한 `Context` 클래스를 새로 구현한다.

■ JXIO Callbacks: JXIO는 콜백 방식으로 동작하기 때문에, RJ-Netty의 워커 간 통신을 서버/클라이언트 구조로 구현하기 위해서 JXIO의 콜백 인터페이스를 새로 구현한다.

- `ServerPortalCallbacks` 클래스: 워커 송신 스레드의 연결 요청을 수신하면, `ServerSession` 객체를 생성하고 이를 `EventQueueHandler` 객체에 등록하여 해당 연결에서 발생하는 메시지 수신 이벤트를 받을 수 있도록 한다.
- `ServerSessionCallbacks` 클래스: 워커 송신 스레드로부터 전송된 메시지를 `MessageDecoder` 객체를 통해 `TaskMessage` 객체의 리스트로 재조립하고, 이를 익스큐터의 입력 큐에 저장한다.
- `ClientSessionCallbacks` 클래스: 워커 수신 스레드의 응답을 수신한다. 즉, 워커 수신 스레드에 보낸 연결 요청 및 메시지 전송에 대한 응답을 수신하고, 이에 대한 메타데이터를 기록한다.
- `EventQueueHandlerCallbacks` 클래스: 워커 수신 스레드에서 메시지를 수신하기 위한 메모리가 부족할 경우 추가로 메모리를 할당한다.

하지만, 이러한 RJ-Netty 서버/클라이언트 구조는 `EventQueueHandler` 객체에 심각한 오버헤드가 발생할 수 있다. 이는 오직 하나의 `EventQueueHandler` 객체에서 모든 이벤트를 수신하여, 콜백을 통해 `ServerSession` 객체 및 `ServerPortal` 객체에게 전달하기 때문이다. 이에 제4.2절에서는 `ServerPortalHandler` 클래스와 `ServerSessionHandler` 클래스를 추가하여, 이 같은 오버헤드 문제를 해결한다.

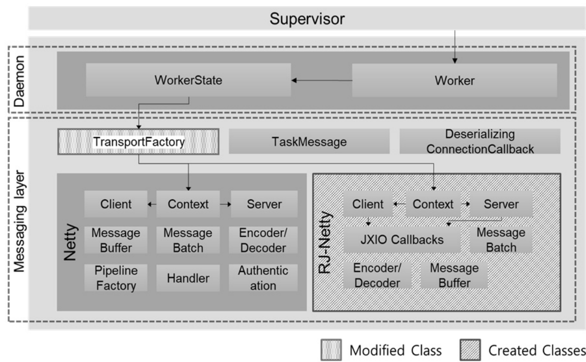


Fig. 7. Structure of RJ-Netty for Worker Communications

4.2 RJ-Netty의 최적화

본 절에서는 제안한 RJ-Netty의 문제점을 분석하고, 이를 해결하기 위해 추가 및 변경한 클래스를 설명한다. Fig. 8은 RJ-Netty 서버의 동작 구조를 나타낸 것으로, 이에 대한 설명은 다음과 같다.

① RJ-Netty 서버는 하나의 ServerPortal 객체와 이에 등록된 하나의 EventQueueHandler 객체를 통해 모든 요청을 수신하고, 이를 ServerPortal 객체와 ServerSession 객체에 전달한다.

② RJ-Netty 클라이언트가 연결을 요청하면, ServerPortal 객체에 등록되어 있는 콜백이 EventQueueHandler 객체를 통해 호출된다. 호출된 콜백은 해당 연결을 관리할 ServerSession 객체를 생성하여 ServerPortal 객체에 등록한다. 이로써, ServerSession 객체에 등록된 콜백이 EventQueueHandler 객체를 통해 호출될 수 있다.

③ EventQueueHandler 객체가 RJ-Netty 클라이언트로부터 메시지 전송을 수신하면, 해당 연결을 관리하는 ServerSession 객체를 찾아 콜백을 호출함으로써 메시지를 처리한다.

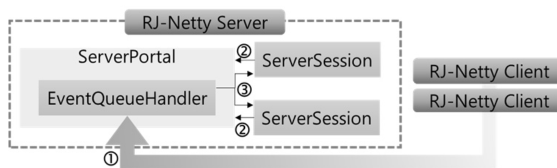


Fig. 8. Working Procedure of the Basic RJ-Netty Server

그러나, 이러한 RJ-Netty 서버 구조는 하나의 EventQueueHadler 객체에서 수많은 이벤트를 처리하기 때문에, EventQueueHandler 객체에 많은 오버헤드가 발생한다. 즉, 모든 수신 이벤트를 ServerSession 및 ServerPortal 객체에 전달하는 과정에서 EventQueueHandler 객체에 심한 오버헤드가 생기는 것이다. 본 논문에서는 이의 해결을 위해 RJ-Netty 서버 구조를 Fig. 9와 같이 개선한다.

① Fig. 8과 마찬가지로, 클라이언트의 연결 요청은 ServerPortal 객체와 이에 등록된 EventQueueHandler 객체에 의해 처리된다. 하지만, 클라이언트가 전송한 메시지는 ServerPortalHandler 및 ServerSession 객체에 의해 처리한다. 즉, ServerSessionHandler 객체에서 ServerSession 객체를 생성하여 ServerPortalHandler 객체의 ServerPortal 객체에 등록한다. 이로써, EventQueueHandler 객체와 ServerSession 객체의 관계가 1:N이 아닌 M:N이 되어, EventQueueHandler 객체의 오버헤드가 크게 감소한다.

② Fig. 8과 동일한 작업을 수행한다. 하지만, 그림 8과 같이 하나의 ServerPortal 객체에 ServerSession 객체를 등록하는 것이 아니라, 여러 개의 ServerPortal 객체에 균등하게 ServerSession 객체를 등록한다.

③ 각 EventQueueHandler 객체가 클라이언트로부터 메시지 전송을 수신하면, 해당 연결을 관리하는 ServerSession 객체를 찾아 콜백을 호출함으로써 메시지를 처리한다.

이와 같은 개선을 통해, 멀티 쓰레드 환경인 Storm에서 발생하는 EventQueueHandler 객체의 오버헤드를 크게 감소시킬 수 있다. 또한, EventQueueHandler 객체의 오버헤드가 감소함으로써 RDMA 기반 Storm의 메시지 처리 성능을 크게 향상시킬 수 있다.

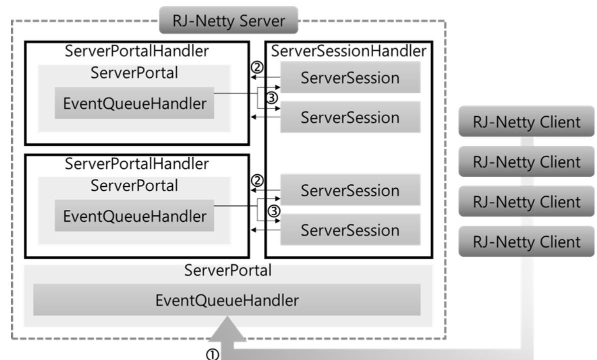


Fig. 9. Working Procedure of the Advanced RJ-Netty Server

5. 실험 평가

본 절에서는 제안하는 RDMA 기반 Storm이 CPU 과부하 문제를 해결하고 메시지 처리 성능을 개선함을 확인하기 위해, IPoIB 및 1GbitE 기반 Storm과의 성능 평가를 진행한다. 제5.1절에서는 실험 환경과 실험 계획을 설명한다. 제5.2절에서는 RDMA, IPoIB, 1GbitE 기반 Storm에 대한 실험 결과를 설명한다.

5.1 실험 환경

실험에 사용된 시스템의 하드웨어 사양은 Table 2와 같다. 또한, 소프트웨어 환경은 운영체제로 CentOS 7을 설치하고, Storm-2.0.0-SNAPSHOT, Zookeeper-3.4.8, JXIO-1.0.3을 각각의 노드에 구축한다. 여기서, Storm의 님버스 데몬은 마스터 노드에서 구동하고, 슈퍼바이저 데몬은 각각의 슬레이브 노드에서 구동한다. 마지막으로, 모든 노드에 주키퍼를 구동시킴으로써 님버스와 슈퍼바이저가 통신하여 상태정보를 공유할 수 있도록 한다.

Table 2. Hardware Specification for the Experiments

Device	Specification	Remark
Master Node	■ Intel E5- 2630V3 2.4GHz 8 Core (Hyper-threading 16 Core) ■ 16G DDR4- 17000 REG ECC RAM * 2 ■ 256G SSD 850 PRO, 1TB HDD	x1
Slave Node	■ Intel E5- 2620V3 2.4GHz 6 Core (Hyper-threading 12 Core) ■ 16G DDR4- 17000 REG ECC RAM * 2 ■ 256G SSD 850 PRO, 1TB HDD	x8
Ethernet	■ 1Gbit LAN on Board	Each node
InfiniBand	■ Mellanox SwitchX®-2 MSX6012F-1BFS Managed FDR 56Gbits/s	Each node

제3절에서 언급한 바와 같이, CPU 과부하 문제와 메시지 처리 성능에 영향을 주는 조건은 Storm의 병렬성과 메시지의 크기이다. 따라서, 본 실험에서는 워커, 스파우트, 볼트의 병렬성과 전송하는 메시지의 크기를 다양하게 변화시키며 실험을 진행한다. 먼저, 병렬성 측면에서는 저병렬성, 중병렬성, 고병렬성의 세 가지 실험 케이스를 구성한다.

■ 저병렬성 실험(Fig. 10): 노드 당 한 개의 워커, 워커 당 한 개의 태스크를 실행한다. 즉, 한 노드에는 워커 한 개가 실행되고, 한 워커에는 스파우트 혹은 볼트 태스크 한 개가 실행된다.

■ 중병렬성 실험: 노드 당 세 개의 워커, 워커 당 세 개의 태스크를 실행한다. 즉, 한 노드에는 워커 세 개가 실행되고, 각 워커는 스파우트 한 개와 볼트 두 개가 실행된다.

■ 고병렬성 실험(Fig. 11): 노드 당 여섯 개의 워커, 워커 당 여섯 개의 태스크를 실행한다. 즉, 한 노드에는 워커가 여섯 개가 실행되고, 각 워커는 스파우트 한 개와 볼트 다섯 개가 실행된다.

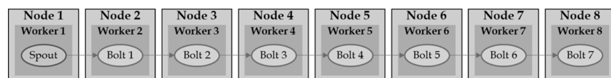


Fig. 10. Configuration of the Low Parallelism Experiment

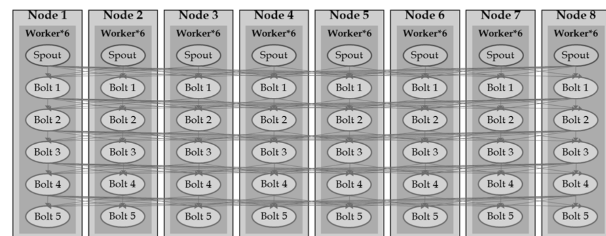


Fig. 11. Configuration of the High Parallelism Experiment

이와 같이 Storm의 병렬성을 세 가지로 구분하고, 메시지 크기를 변화시키며 다음 두 가지 항목을 평가한다.

■ 메시지 처리량(message throughput): 임의의 시간 동안 Storm에서 처리한 메시지 수 또는 메시지 크기를 측정한다. 본 실험에서는 Storm에서 제공하는 쓰리프트 포트를 통해 10분 동안 Storm에서 처리한 메시지의 수를 측정하고, 이를 MB/s 단위로 환산하여 처리량을 계산한다.

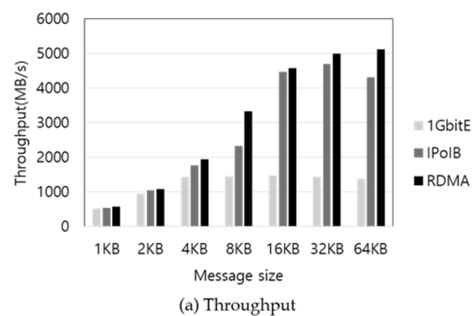
■ CPU 부하 평균(load average): Storm에서 토폴로지를 처리할 때 발생하는 CPU 부하 정도를 측정한다. 본 실험에서는 리눅스 top 명령어를 통해 각각의 슈퍼바이저 노드에 대한 CPU 부하를 수집하고, 이의 평균을 측정한다.

또한, 실험을 위해 Storm의 토폴로지를 다음과 같이 설계한다. 먼저, 토폴로지는 각 실험에 해당하는 만큼 워커, 스파우트, 볼트를 생성하고, 스파우트와 여러 볼트들을 셔플 그룹핑으로 묶는다. 여기서, 셔플 그룹핑을 사용한 이유는 태스크 간에 랜덤하게 메시지를 전송함으로써, 워커 간 통신을 최대한 많이 발생시키기 위함이다. 다음으로, 스파우트는 초기화 단계에서 실험에 필요한 메시지 크기만큼 튜플을 생성하고, 이를 다음 볼트에게 전송한다. 마지막으로, 볼트는 스파우트 혹은 다른 볼트로부터 수신한 튜플을 바로 다음 볼트로 전송한다. 이로써, 각 통신 프로토콜 따른 Storm의 메시지 성능을 측정할 수 있다.

5.2 실험 결과

1) 저병렬성 실험

먼저, Fig. 12는 1GbitE, IPoIB, RDMA 기반 Storm에 대한 저병렬성 실험 결과를 나타낸다. 첫 번째로, Fig. 12(a)는 메시지 처리량에 관한 실험으로, RDMA의 메시지 처리량이 가장 높게 나타났고, 이어서 IPoIB, 1GbitE 순인 것을 알 수 있다. 이와 같은 결과가 나타난 이유는 대역폭이 1Gbits로 제한된 1GbitE 장비와 다르게 InfiniBand는 56Gbits로 높은 대역폭을 지원하기 때문이다. 특히, 메시지 크기가 클수록 그 차이가 큰 데, 이는 InfiniBand의 높은 대역폭이 대용량 메시지에서 효과가 큼을 의



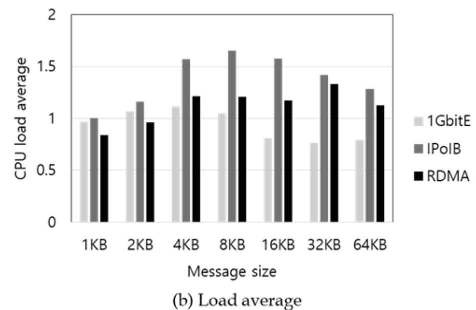


Fig. 12. Experimental Results on Low Parallelism

미한다. 또한, 운영체제 오버헤드를 줄이는 RDMA가 기존 IPoIB 보다도 높은 처리량을 보임을 확인할 수 있다.

두 번째로, Fig. 12(b)는 CPU 부하 평균 실험 결과를 나타낸다. 그림을 보면, 대역폭이 높아 더 많은 메시지를 처리하는 IPoIB가 1GbitE 대비 더 많은 CPU 자원 사용이 있음을 알 수 있다. 즉, IPoIB 기반 Storm이 메시지 처리 성능을 향상시켰으나 잦은 문맥 전환과 버퍼 복사로 인해 CPU 사용률이 증가했음을 보여준다. 반면에, RDMA는 운영체제 개입 없이 메시지를 전송하여 IPoIB에 비해 메시지 처리량을 증가시켰음에도 불구하고 CPU 사용률은 오히려 감소시킨 것으로 나타났다.

2) 중병렬성 실험

다음으로, Fig. 13은 중병렬성 실험 케이스에 대한 결과를 나타낸다. 첫 번째로, Fig. 13(a)는 메시지 처리량에 관한 실험으로, 저병렬성 실험 대비 병렬성이 3배 증가함에 따라, 스파우트는 더 많은 메시지를 생성하고 볼트는 더 많은 메시지를 처리함으로써, 전체적인 처리량이 크게 증가하였음을 확인할 수 있다. 또한, 제안하는 RDMA 기반 Storm의 메시지 처리량이 최대 14,800MB/s로 가장 우수함을 볼 수 있다. 반면에, 1GbitE은 대역폭의 한계로 인해, 최대 1,650MB/s의 처리량에 머물며, IPoIB는 최대 12,500MB/s로 RDMA에 미치지 못함을 볼 수 있다. Fig. 13(a)의 중병렬성 실험 결과를 요약하면, RDMA는 IPoIB에 비해 최대 1.22배까지, 1GbitE에 비해 최대 10.2배까지 메시지 처리량을 개선한 것으로 나타났다.

두 번째로, Fig. 13(b)는 중병렬성 실험에서 각 노드의 CPU 부하 평균을 나타낸다. 그림을 보면, 1GbitE는 대역폭의 한계로 인해 네트워크를 통한 데이터 전송 시간이 길어서 상대적으로 CPU에 부하가 덜한 것을 알 수 있다. 반면에, 높은 대역폭을 지원하는 IPoIB는 문맥전환과 버퍼 복사가 자주 발생하고, 이로 인해 CPU 부하 평균의 수치가 최대 10까지 오르는 것을 확인할 수 있다. 하지만, RDMA의 경우, 이러한 CPU 부하를 크게 감소시켜 부하 평균 수치가 IPoIB의 절반인 최대 5로 나타나는 것을 볼 수 있다. Fig. 13의 실험 결과를 보면, RDMA는 IPoIB 보다 메시지 처리량을 늘렸음에도 불구하고, CPU 부하는 최대 2.0배 개선한 것으로 나타났다.

3) 고병렬성 실험

마지막으로, Fig. 14는 고병렬성 실험을 수행한 결과이다. 첫 번째로, Fig. 14(a)는 메시지 처리량에 관한 실험으로, 저병렬성 실험 대비 병렬성이 6배 증가함에 따라, 이전보다 더 많은 메시지를 생성한다. 하지만, 1GbitE와 IPoIB는 중병렬성 실험과 마찬가지로 처리량에 큰 변화가 없었고, RDMA의 처리량도 소폭 증가하는데 그쳤다. 이는 중병렬성 실험보다 고병렬성 실험에 구성한 볼트의 단계가 더 깊어서, 메시지가 마지막 볼트까지 도달하는데 더 오랜 시간이 걸리기 때문에 나타나는 현상이다. 즉, Fig. 14에서 보듯이 고병렬성 실험에서 구성한 볼트들의 레이어가 과도하게 깊어져 전체적인 메시지 처리량은 큰 변화가 없는 것이다. 그렇지만, 고병렬성 실험에서도 여전히 RDMA가 다른 두 통신 방법에 비해 우수한 메시지 처리량을 보임을 확인할 수 있다. Fig. 14(a)의 고병렬성 실험 결과를 요약하면, RDMA는 IPoIB에 비해 최대 1.25배, 1GbitE에 비해 최대 9.30배까지 메시지 처리량을 개선한 것으로 나타났다.

두 번째로, Fig. 14(b)는 고병렬성 실험에서 CPU 부하 평균을 나타낸다. 그림을 보면, 중병렬성 실험과 마찬가지로 1GbitE는 대역폭의 한계로 인해 데이터 전송 시간이 길어서 CPU 부하가 상대적으로 낮음을 확인할 수 있다. 반면에, 높은 대역폭을 지원하는 IPoIB는 문맥전환과 버퍼 복사가 자주 발생하고, 이로 인해 CPU 부하 평균이 최대 27까지 올라가

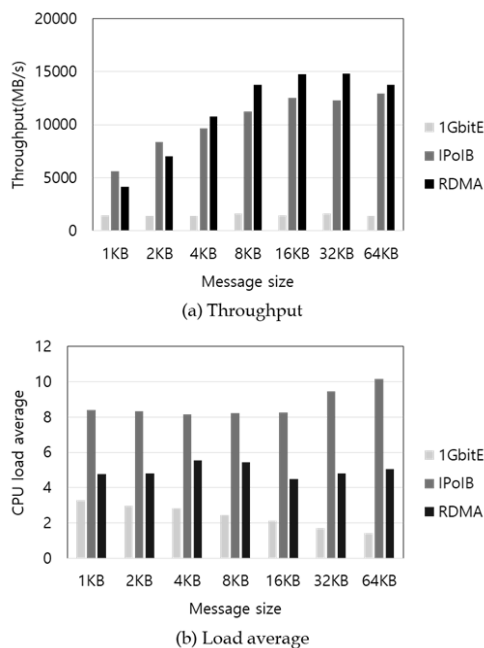


Fig. 13. Experimental Results on Medium Parallelism

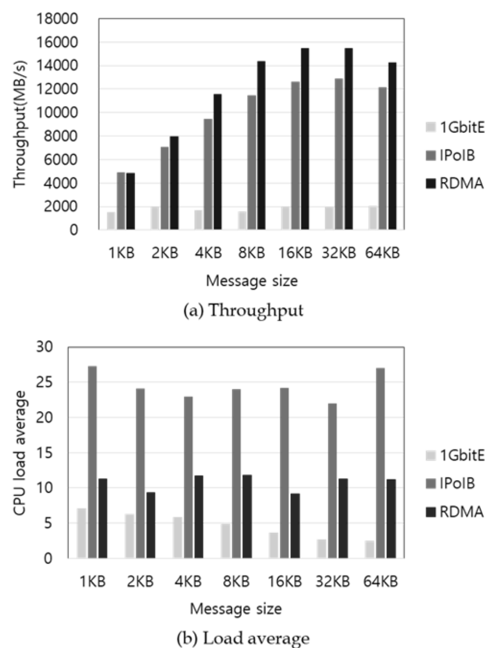


Fig. 14. Experimental Results on High Parallelism

CPU에 과부하가 발생하는 것을 확인할 수 있다. 반면에, RDMA는 이러한 CPU 부하를 감소시켜 부하 평균이 IPoIB의 절반 이하인 최대 11에 불과함을 알 수 있다. Fig. 14의 결과를 보면, RDMA는 IPoIB 대비 메시지 처리량과 지연시간을 향상시켰음에도 불구하고 CPU 부하는 최대 2.6배까지 개선한 것으로 나타났다.

6. 결 론

본 논문에서는 InfiniBand RDMA를 사용해 Apache Storm의 성능을 크게 향상시키는 방법을 제안하였다. Storm은 분산된 서버에서 다수의 워커 프로세스를 통해 대용량 스트림 데이터를 처리하는 구조로, 이러한 워커 간 통신을 위해 TCP/IP 기반 통신 프레임워크인 Netty를 사용하였다. 이러한 구조에서는 메시지 전송 과정에서 잦은 문맥 전환과 버퍼 복사가 발생하여 많은 CPU 자원을 필요로 한다. 특히, 고성능 네트워크 장비인 InfiniBand에서 IPoIB를 이용하여 Storm의 메시지를 처리할 경우 이러한 CPU 과부하 문제가 더욱 심각해진다. 이 같은 CPU 과부하 문제로 인해 InfiniBand의 성능을 최대로 활용하지 못할 뿐 아니라, 경우에 따라 전체 시스템이 마비될 정도의 심각한 문제를 초래할 수 있다.

본 논문에서는 InfiniBand의 RDMA 기능을 Storm에 적용하여 CPU 자원 요구를 최소화하고 메시지 처리량과 지연시간을 향상시키는 RDMA 기반 Storm을 설계 및 구현하였다. RDMA는 운영체제를 거치지 않고 호스트 메모리에서 원격지 메모리로 직접 데이터를 전송할 수 있어, 고성능 네트워크 장비인 InfiniBand의 성능을 최대로 활용할 수 있는 기술이다. 이를 위해, 먼저 Storm의 네트워크 구조를 분석하여, CPU 과부하 문제의 원인을 도출하였다. 다음으로, 기존 Netty를 대신할 RDMA 기반의 새로운 네트워크 전송 계층인 RJ-Netty 구조를 설계 및 구현하였다. 그리고, Storm에서의 메시지 처리량을 최대화할 수 있도록 JXIO 서버 구조를 최적화하였다. 마지막으로, 다양한 실험을 통해 제안하는 RDMA 기반 Storm이 기존 1GbitE 및 IPoIB 기반 Storm에 비해 우수함을 입증하였다. 실험 결과, RDMA를 Storm에 적용함으로써, IPoIB에서 발생하던 CPU 과부하 문제를 해결할 수 있을 뿐만 아니라 메시지 처리량을 향상시킬 수 있었다. 본 논문은 InfiniBand의 RDMA 기능을 Storm에 적용한 최초의 시도로써, InfiniBand의 높은 성능을 Storm에서 최대로 활용할 수 있는 우수한 연구 결과라 사료된다.

References

[1] Apache Hadoop [Internet], http://hadoop.apache.org/.

[2] Apache Storm [Internet], http://storm.apache.org/.

[3] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, “Storm@Twitter,” In *Proc. of the Int'l Conf. on Management of Data*, ACM SIGMOD, Snowbird, Utah, pp.147-156, Jun. 2014.

[4] P. Goetz and B. O'Neill, Storm Blueprints: Patterns for Distributed Real-time Computation, Packt Publishing, Mar. 2014.

[5] Apache Spark [Internet], https://spark.apache.org/.

[6] Apache S4 [Internet], http://incubator.apache.org/projects/s4.html/.

[7] Apache Flink [Internet], https://flink.apache.org/.

[8] Infiniband Trade Association [Internet], http://www.infinibanddta.org/.

[9] X. Lu, N. S. Islam, M. Wasi-Ur-Rahman, J. Jose, H. Subramoni, H. Wang, and D. K. Panda, “High-performance Design of Hadoop RPC with RDMA over InfiniBand,” In *Proc. of the IEEE 42nd Int'l Conf. on Parallel Processing (ICPP)*, Lyon, France, pp.641-650, Oct. 2013.

[10] X. Lu, M. Wasi-Ur-Rahman, N. Islam, D. Shankar, and D. K. Panda, “Accelerating Spark with RDMA for Big Data Processing: Early Experiences,” In *Proc. of the IEEE 22nd Annual Symp. on High-performance Interconnects*, Mountain View, CA, pp.9-16, Aug. 2014.

[11] J. Huang, X. Ouyang, J. Jose, M. Wasi-Ur-Rahman, H. Wang, M. Luo, H. Subramoni, C. Murthy, and D. K. Panda, “High-performance Design of HBase with RDMA over InfiniBand,” In *Proc. of the IEEE 26th Int'l Parallel and Distributed Processing Symp. (IPDPS)*, Shanghai, China, pp. 774-785, May 2012.

[12] M. Wasi-Ur-Rahman, N. S. Islam, X. Lu, J. Jose, H. Subramoni, H. Wang, and D. K. Panda, “High-performance RDMA-based Design of Hadoop MapReduce over InfiniBand,” In *Proc. of the IEEE 27th Int'l Parallel and Distributed Processing Symp. Workshops (IPDPSW)*, Cambridge, MA, pp. 1908-1917, May 2013.

[13] N. S. Islam, M. Wasi-Ur-Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda, “High Performance RDMA-based Design of HDFS over InfiniBand,” In *Proc. of the Int'l Conf. on High Performance Computing, Networking, Storage, and Analysis*, Salt Lake City, UT, pp.1-12, Nov. 2012.

[14] Netty [Internet], https://netty.io/.

[15] Context Switch [Internet], https://en.wikipedia.org/wiki/Context_switch/.

[16] JXIO [Internet], https://github.com/accelio/JXIO/.

[17] Apache Zookeeper [Internet], http://zookeeper.apache.org/.

[18] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, “Zookeeper: Wait-free Coordination for Internet-scale Systems,” In *Proc. of the USENIX Annual Technical Conf.*, Boston, MA, pp.1-6, Jun. 2010.

[19] Apache Thrift [Internet], https://thrift.apache.org/.

[20] Accelio (Official website) [Internet], http://www.accelio.org/.

[21] Accelio (Open source) [Internet], https://github.com/accelio/accelio/.

[22] Lmax disruptor [Internet], https://lmax-exchange.github.io/disruptor/.

양 석 우
https://orcid.org/0000-0003-0518-1974
e-mail : seokwoo@kangwon.ac.kr
2016년 강원대학교 컴퓨터과학과(학사)
2018년 강원대학교 컴퓨터과학과(석사)
관심분야 : 빅데이터, 데이터스트림, 하둡 에코시스템

손 시 운
https://orcid.org/0000-0001-5647-7527
e-mail : ssw5176@kangwon.ac.kr
2014년 강원대학교 컴퓨터과학과(학사)
2016년 강원대학교 컴퓨터과학과(석사)
2016년～현 재 강원대학교 컴퓨터과학과 박사과정
관심분야 : 데이터마이닝, 빅데이터, 하둡 에코시스템

문 양 세
https://orcid.org/0000-0002-2396-0405
e-mail : ysmoon@kangwon.ac.kr
1991년 한국과학기술원 전산학과(학사)
1993년 한국과학기술원 전산학과(석사)
2001년 한국과학기술원 전산학과(박사)
1993년～1997년 현대전자산업(주) 주임연구원
2001년～2002년 (주)현대시스콤 선임연구원
2002년～2005년 (주)인프라밸리 기술위원(이사)
2005년～2008년 한국과학기술원 첨단정보기술연구센터 연구원
2008년～2009년 미국 퍼듀대학교 방문연구원
2012년～2013년 강원대학교 기획부처장
2014년～2016년 강원대학교 IT대학 부학장
2005년～현 재 강원대학교 컴퓨터과학과 교수
관심분야 : 데이터마이닝, 스트림데이터, 저장 시스템, 데이터베이스 응용, 빅데이터 분석, 프라이버시 보호 마이닝