Distributed Queue를 이용한 서버 가용성 향상 연구

박봉희 ○ 최진영
고려대학교 컴퓨터정보통신대학원
pbhee@korea.ac.kr, choi@formal.korea.ac.kr

A Case Study on Improve the availability of the Server Using the Distributed Queue.

Bonghee Park○, Jinyoung Choi
Korea University

요 약

기업에서 사용하는 시스템에서는 대량의 데이터를 실시간으로 처리해야 하며 또한 시스템 장애로 인한 생산 중단이 발생하지 않도록 하기 위해 서버 이중화 구성에 많은 투자를 하고 있다. 장애를 최소화하기 위해 서버 구성을 Active-Standby 구조로 이중화를 구성하여 사용하고 있으며, 이로 인해 시스템에 대한 많은 비용을 투자하고 있다. 서버 부하 감소 및 서버 가용성 향상을 통한 투자 비용을 절감하기 위해 서버 분산 처리에 대한 방법이 나오고 있다, 그러나 Network 스위치나 라우터 등의 Hardware 적인 분산 시스템을 이용한 방법으로는 특정한 데이터만을 분산하여 처리할 수 없고 Hardware의 투자 비용 또한 증가하게 된다. 또한 서버의 Active-Standby 구성에서 Standby 서버는 Active 서버의 문제 발생시에만 이용하게 됨으로 인해 서버의 가용성이 떨어지는 불합리가 발생함으로 인해 이를 개선하기 위한 연구를 통해 Active-Active 구성을 통해 서버의 가용성을 확보하고자 한다.

본 논문에서는 서버 가용성 향상을 위한 방법으로 DQ(Distributed Queue)를 이용하여 응용프로그램에서의 데이터 분산을 통한 응용프로그램에서의 부하를 분산하여 서버 내 리소스를 최대한 활용하면서 응용프로그램에서의 부하를 감소화할 수 있는 방법을 검증하고자 한다.

1. 서 론

기업에서 사용하는 시스템의 운영은 서버(Server)와 클라이언트(Client) 그리고 데이터베이스 (Database)로 구성되어 사용되고 있다. 특히 생산 관련된 업무를 하는 생산시스템에서는 장애 방지를 위해 시스템의 이중화 구성에 많은 비용을 투자하고 있으며, 생산 설비에서 나오는 데이터(Data)양의 증가로 시스템의 대형화 및 안정적인 운영에 대한 비용 증가의 문제를 가지게 된다. 생산시스템에서는 시스템 장애를 예방하기 위해 동일한 사양의 Hardware를 이중화로 구성하여 Active-Standby 구조로 운영하고 있다. 이러한 구조는 Active 서버에서 대부분의 업무를 처리하고 Standby 서버는 업무를 하지 않고 있어 이중 투자에 대한 비용 부담을 가지게 된다.

따라서 이와 같은 구조에서 높은 가용성을 유지하기 위해 SBL(Server Load Balancer)를 이용한 부하 분산 방안을 많이 사용하고 있다. SLB는 클라이언트로부터 요청되는 세션을 서버상태를 고려하여 적절하게 분해하는 역할을 수행하며 SLB를 사용함으로써 서비스는 유연성과 가용성 그리고 확장성을 높일 수 있게 되었다[1]. 초기 부하분산 처리에는 DNS(Domain Name Service)기반의 방법이 시도되었으나 IP 주소와 caching 문제로 효과적인 부하분산 처리를 할 수 없었다[2].

그러나 이러한 분산 방법은 데이터를 모두 Network Level 에서 분산하도록 되어 있어, 특정한 포트나 미리 정의된 데이터만 응용 프로그램으로 전달하지 못하는 문제가 있다. 서버의 가용성 향상을 위해서 서버의 Active-Active 기능을 응용프로그램에서 구현해야 하는데, 이는 특정한 데이터만 응용프로그램에서 처리하도록 해야 하는 것이다.

본 논문에서는 응용프로그램을 수정하지 않고 Tibco에서 제공하는 DQ(Distributed Queue)를 이용하여 서버 Load Balancing를 통해 가용성을 향상하는 방안을 연구하고 결과를 확인해 보고자 한다[3].

2. 관련 연구

프로세스 사이의 Load Balancing 구성에 대한 기존 시스템은 별도의 Load Balancer 서버를 이용하여 서버간의 데이터를 전체적으로 Load Balancing 하는 것으로 서버에서는 하나의 프로그램만 실행하여야 하는 문제가 있으며, Load Balancer 서버에서 Load Balancer 역할을 구현하여 운영해야 하는 문제가 있다.

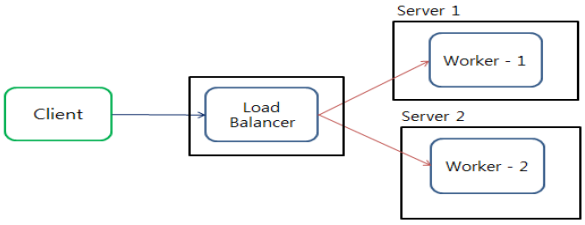


그림 1. 기존 Load Balancing 구성

이와 같은 문제를 해결하기 위해 서버가 아닌 응용프로그램에서 메시지 Load Balancer 기능을 하는 Rendezvous Distributed Queue를 통해 Load Balancer 기능을 구현하며 또한 서버에서 동일한 기능을 하는 응용프로그램을 여러 개를 실행할 수 있도록 한다. Load Balancer 기능을 하는 Scheduler도 Worker와 동일한 기능을 실행하는 응용프로그램으로 config를 통해 역할을 할당한다. 또한 응용프로그램을 여러 개 실행하여 한 개의 Scheduler와 여러 개의 Worker를 사용한다.

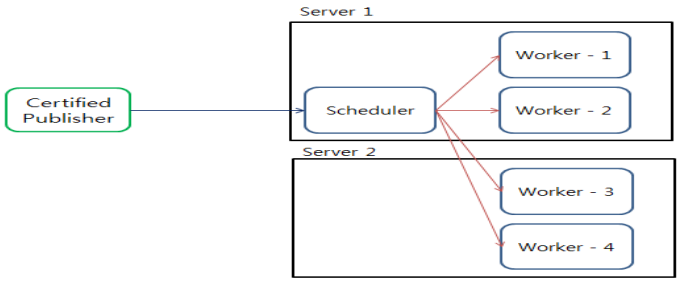


그림 2. DQ를 이용한 Load Balancing 구성

Rendezvous 통신 모드 중 DQ(Distributed Queue) 기능은 CM(Certified Message) 메시지의 Load Balancing 기능을 구현하며 Scheduler는 FT(Fault-Tolerance) 요소로서 각 DQ에 대해 정확히 하나만 존재해야 한다. 또한 Scheduler는 자신도 필요 시 메시지를 처리한다. DQ를 사용하기 위해 Rendezvous의 TibrvCmTransport 구현하여야 하며 CM Name을 DQ group에서 동일해야 한다. Weight이라는 config 설정을 통해 Scheduler와 worker를 구분하여 응용프로그램의 역할을 정의한다. Scheduler와 Worker Member 모두 동일한 기능의 응용프로그램을 실행하는 것으로 CM Name을 모두 하나의 값으로 하고, Scheduler Member만 weight 값을 높게 설정하고, 나머지 Worker Member는 weight값을 동일하게 설정하여 Worker의 역할을 분산하여 처리되도록 한다. Scheduler는 Worker에게 골고루 메시지를 할당하는 역할을 한다. Scheduler와 Worker는 내부 통신을 통해 상태를 공유하며 놀고 있는 Worker에게 Scheduler가 메시지를 할당하여 처리하도록 한다. Worker Member는 전체 메시지를 수신하지만 Scheduler에 의해 처리하는 역할을 할당 받은 Worker만이 해당 메시지를 처리한다.

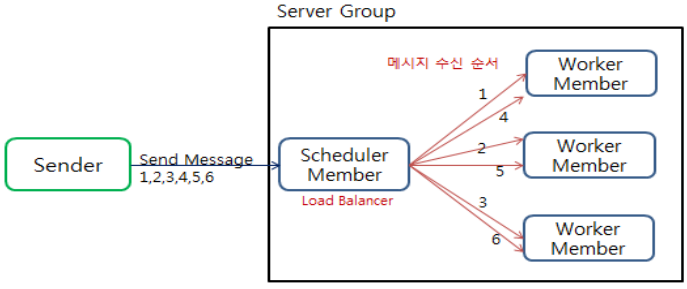


그림 3. DQ 구성 및 메시지 처리 순서

서로 다른 서버에서 데이터의 분산을 순서대로 처리하는 기능을 통해 하나의 서버에서 프로세스별 데이터 분산처리를 하거나 서로 다른 서버에서 동일한 종류의 데이터를 분산처리를 할 수 있는 기능을 구현한다. 이번 연구는 Load Balancing 기법을 네트워크 영역에서 하는 것이 아니라 응용프로그램에서 실시함으로 인해 Hardware에 제한되지 않고 Load Balancing를 통해 서버 가용성을 향상시키는 것이다

3. 본 론

서로 다른 역할을 하는 서버와의 데이터의 처리를 위해 각 네트워크 설정을 다르게 하여 데이터의 종류에 따라 다른 응용프로그램으로 데이터가 전달이 되도록 함으로서 응용프로그램에서는 해당하는 데이터만 처리함으로 인해 서버 가용성이 향상되도록 하는 것이다.

본 연구에서는 특정 시스템에서 데이터를 전달받아 서버 내 응용프로그램으로 데이터가 분배되는지 확인하고, 데이터가 처리될 때 성능의 분배가 되는지 검증하였다.

3.1 시스템 구성

시스템간의 데이터 전달에 대한 구성으로 외부 다른 시스템으로부터 전달받아 분배하여 응용프로그램에서 데이터를 처리한 후 다시 Host라는 또 다른 역할을 하는 서버로 관련된 데이터를 전송하는 구조로 서로 다른 역할을 하는 서버간에 데이터를 전달하게 된다.

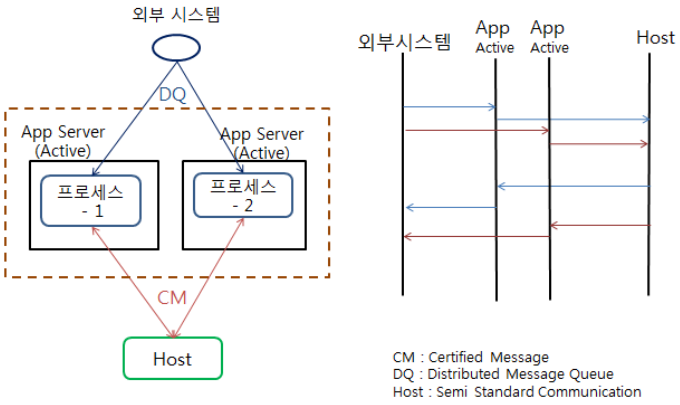


그림 4. 시스템간의 구성

두 개 이상의 서로 다른 서버에서 동일한 응용프로그램을 실행하고 각 응용프로그램에서는 외부에서 전달되는 데이터를 분산하여 서버의 가용성을 향상할 수 있고 동시에 처리해야 하는 데이터에 대해 부하를 감소할 수 있도록 하는 것을 검증하는 것이다.

3.2 실험 환경

본 연구에서는 응용프로그램에서의 데이터 분산 여부를 확인하기 위해서 다음과 같은 응용프로그램을 이용한다. 외부시스템에서 전송할 데이터를 생성하여 Tibco의 CM(Certified Message)를 이용하여 전송하는

프로그램과 외부로부터 데이터를 분산하여 전달받을 수 있도록 하는 Tibco의 DQ를 이용한 메시지를 수신할 수 있는 응용프로그램을 실행함으로서 데이터가 순서대로 분할이 되는지 확인하며 이 때 CPU의 성능이 분산되는지 검증한다 . 다음과 같이 서버에서 응용프로그램을 1개 ,2개를 각각 실행하여 성능에 대한 리소스 변화를 측정한다.

응용프로그램		비교분석요소
서버	1개 실행 2개 실행	CPU, Thread

표 1. 비교분석 요소

검증을 위한 시스템 환경은 다음과 같이 윈도우 시스템을 이용하여 동일한 네트워크 대역을 이용한다.

클라이언트	OS	CPU	Memory	HDD
	Windows 7	2.4 GHz	4 GB	250 GB
측정프로그램	윈도우 성능 측정 프로그램			

표 2. 클라이언트 시스템 환경

3.3 데이터 분산 시험 실시

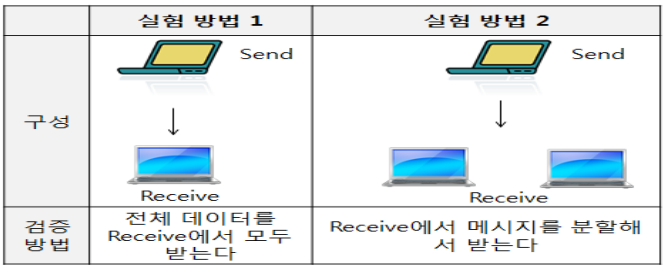
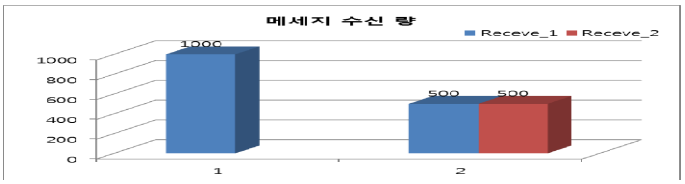


그림 5. 데이터 분산 시험 구성

데이터 분산이 이루어지는지 확인하기 위해 Send 프로그램을 통해 데이터를 만들어 전송하고 Receive 프로그램을 실행하여 정상적으로 받는지 확인한다. Send에서는 데이터를 순차적으로 생성하는데, 1부터 차례로 증가하는 데이터를 만들어 전송하고 receive 프로그램 하나일 경우에 모든 데이터를 받는지 검증하고, receive 프로그램 2개 일 때와 3개 일 때 데이터가 골고루 분배되어 전송이 되는지 확인한다. 그리고 이 때 리소스 측정 프로그램을 통해 각 receive 프로그램의 CPU 성능을 확인한다.

3.4 데이터 분산 시험 결과 검증

데이터 분산이 정상적으로 되는지 프로그램을 통해 확인한 결과 receive 프로그램 1개 일 경우 send한 전체 데이터를 모두 정상적으로 받았으며, receive 프로그램 2개 일 때는 2개로 순차적으로 분산되어 전송되는 것을 확인할 수 있었다. Send 프로그램을 통해 100개의 데이터를 만들어 전송하였으며, Receive 프로그램에서는 전송 받은 데이터를 화면에 표시하는 기능을 구현하여 확인한 결과 프로그램 두 개의 경우 500개씩 데이터가 분산되어 수신됨을 확인하였다.



그래프 1. 데이터가 절반으로 분산되어 수신

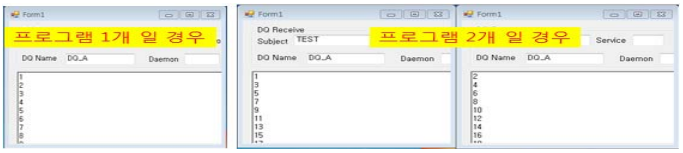
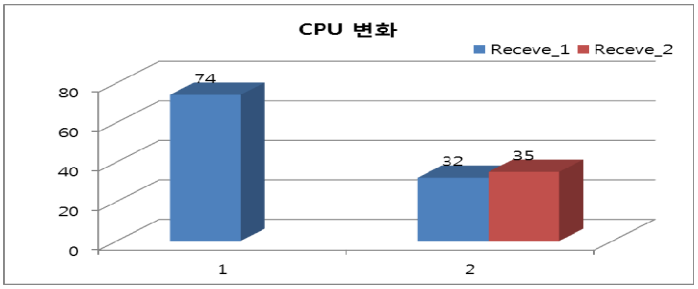


그림 6. 데이터가 정상적으로 분산되는지 변화

리소스 변화에 대한 검증 방법으로 Receive 프로그램 1개일 경우와 2개 실행할 경우에 대해 CPU 성능을 확인하는 방법을 사용하였다. 프로그램 1개일 경우 74%에서 2개일 경우 32% 수준으로 절반의 CPU의 사용량을 확인하였다.



그래프 2. 데이터가 절반으로 분산되어 수신

그림 7. 프로그램 분산에 따른 CPU 분산 변화

4. 결 론

서버 가용성 향상을 위한 서버 분산에 대한 방법으로 라우터 등과 같은 장치를 이용하고 있으나, 동일한 Network 설정에서 특정 데이터만을 분산이 어려움이 있다. 이에 Network 부하까지 고려하여 분산할 수 있도록 하는 기술인 DQ(Distributed Queue)를 이용하여 응용프로그램에서 데이터 분산을 하는지를 데이터 수신 프로그램을 통해 검증하였다. 이를 이용하여 하나의 서버 내에서 또는 물리적으로 분리된 서로 다른 서버에서도 응용프로그램간에 데이터 분산이 정확히

이루어지는 것을 확인하였으며, 또한 서로 다른 서버에서 리소스 분산이 이루어지면서 응용프로그램의 부하가 감소하는 것을 확인하였다.

이 연구를 통해 DQ(Distributed Queue)를 사용하여 데이터 특성에 맞게 Network 통신 환경을 구성하고 응용프로그램에서의 데이터 분산 처리를 통한 서버 부하 감소를 통해 서버 성능에 대한 가용성 향상을 할 수 있다는 것을 검증하였다.

참고문헌

[1] Matthew Syme, “Optimizing Network Pervormance with Content Switching: Server, Frewall and Cache Load Balancing”, Prentice Hall, 2004

[2] Pual Albitz, “DNS and Bind”, Fifth Ed., oreilly & Asssociates Inc, 2006

[3] tibco “Distributed Queue in TIBCO Rendezvous Concepts”

[4] tibco “TIBCO Rendezvous Concepts”