AOSA의 내용중 ZeroMQ에 대한 내용 요약

http://www.aosabook.org/en/zeromq.html

ZeroMQ는 messaging system이다. 처음에는 엄청 빠른 messaging system을 만드는 것으로 시작하였다. 그래서 처음 1년은 벤치마킹 방법을 고안하고 가능한한 효율적인 아키텍쳐를 만드는데 주력하였다. 2년째에는 distributed application을 만들고 임의의 message pattern이나 다양한 전송 방법이나 여러 language binding을 지원하는데 초점을 맞추었다. 3년째에는 사용성을 향상시키고 배우기 쉽도록 만드는데 주력하였다.

1. Application vs Library

ZeroMQ는 messaging server가 아닌 library다.
퍼포먼스의 측면에서 볼 때 중간에 서버가 있으면 네트워크를 두번 타게 되는 단점이 생기고 이 서버가 bottleneck의 원인이 될 수 있다.
두번째로 생각해 볼 수 있는 것은 large-scale deployment이다. deployment가 여러 회사에 걸쳐 있게 되면 각각의 회사들은 다른 회사에 서버가 있기를 원하지 않을 것이다. 그러다보니 각각의 회사에 서버가 있게 되고 이것은 관리라든지의 어려움이 생기게 된다. 이것을 해결하려면 각각의 컴포넌트가 별도로 동작하게 하는 완전한 분산 아키텍쳐가 필요하게 된다. 결국 messaging library이다.
ZeroMQ는 서버없이 메시징을 어떻게 할 수 있을까에서 시작하였다. 이에대한 결론이 library이다.
이 아키텍쳐가 더 효율적이고 더 유연하다는 것을 그동안 증명할 수 있었다.
의도하지 않았던 결론중의 하나는 라이브러리 모델이 제품의 사용성을 향상시켰다는 것이다. 사용자들은 서버를 설치하고 운용하지 않아도 된다는 것에 많은 점수를 주었다.

배운점: 처음 프로젝트를 시작할 때 가능하다면 라이브러리의 형태로 만드는 것을 고려해라.

2. Global State

라이브러리에 전역변수는 좋지 않다. 라이브러리는 한 프로세스내에서도 여러번 로드될 수 있는데 이렇다 하더라도 전역변수는 하나만 있게 된다. 그러면 서로 다른 두 군데서 라이브러리의 전역변수는 사용하려고 하면 race condition이나 실패같은 상황이 발생 할 수 있다.
이 무제를 해결하기 위해 ZeroMQ는 전역변수를 사용하지 않는다. 따라서, ZeroMQ를 사용하는 사용자가 필요한 전역변수를 만들어서 사용해야 한다. 이것은 context라고 부르는데 전역변수를 가지고 있는 하나의 object라고 말할 수 있다. 만약 두 군데서 라이브러리를 사용하고 각자 자신의 context를 가지고 사용한다면 서로서로 상대방의 context의 존재를 모를 것이다.

배운점: 라이브러리에서 전역변수를 사용하지 말아라.

3. Performance

메시지 시스템의 속도는 throughput과 latency로 이야기 할 수 있다: throughput - 주어진 시간에 얼마나 많은 메시지가 지나 갈 수 있는가, latency - 메시지가 도착하는데 얼마의 시간이 걸리는가.
여기서 중요하게 이해하고 있어야 하는 점이 있다. latency는 A에서 B로 도착하는 데 각각의 메시지가 걸리는 시간을 의미하는 것이고 throughput은 A에서의 throughput, B에서의 throughput으로 표현되는 것이라는 것이다.

배운점: 직면한 문제를 명확하게 이해하라.

4. Critical Path

아래의 세가지가 퍼포먼스에 중요한 역할을 한다.

- Number of memory allocations
- Number of system calls
- Concurrency model

하지만, 모든 memory allocation이나 system call이 중요하지는 않다. 메시징 시스템에서는주어진 시간

동안 A에서 B로 얼마나 많은 메시지를 전달할 수 있느냐가 중요하다. 또한 한 메시지를 얼마의 시간만에 전달할 수 있느냐도 중요할 수 있다.
매우 자주 사용되는 코드의 부분을 critical path라 부르는데 이 부분이 최적화에 초점이 맞추어져야 하는 부분이다.

배운점: critical path에 있지 않은 부분을 최적화하려 하지 마라.

5. Allocating Memory

message allocation과 message copying을 잘 조절하는 것이 퍼포먼스에 중요하다. 작은 메시지는 copy를 하는 것이 낫고, 큰 메시지는 allocation을 하는 것이 낫다.
ZeroMQ는 handle이라고 하는 것을 사용하는데 작은 메시지의 경우는 handle에 데이터를 저장하고 큰 메시지의 경우는 데이터에 대한 포인터만을 저장한다.

배운점: 퍼포먼스를 생각할 때 한가지만을 생각하지 마라. 각각의 경우(작은 메시지와 큰 메시지)에 다른 해결책이 있을 수 있다.

6. Batching

4개의 메시지를 각각 따로 보낸다면 전체 네트워크 스택을 4번 타야하기 때문에 좋지 않다. 이런 경우 여러 메시지를 한번(batching)에 보낼 수 있다면 좋을 것이다(throughput의 향상). 하지만, 반대로 이것은 latency가 나쁘게 된다.
ZeroMQ는 다음의 방법을 사용한다. 메시지가 많지 않고 네트워크 스택의 bandwidth를 넘지 않으면 latency를 향상시키기 위해 batching을 사용하지 않는다. 네트워크 스택의 bandwidth를 넘으면 메시지를 큐에 저장한다. 이러한 상황에서는 batching을 사용한다.

배운점: throughput을 향상시키기 위해 제일 위의 레이어에서만(아래의 레이어에서의 batching은 의미없다) batching을 사용한다. 메시지가 쌓이는 경우만 batch를 사용한다.

7. Architecture Overview

사용자는 여러 peer와 통신할 수 있는 socket이라는 것으로 ZeroMQ와 통신한다.
socket object가 사용자의 쓰레드에 있고 이 외에 ZeroMQ는 통신의 비동기를 다루기 위해 여러개의 worker 쓰레드를 만든다.(네트워크로부터 데이터 읽기, 메시지 큐에 넣기, 접속 연결 허락하기 등)
worker 쓰레드에는 여러 object가 있는데 이들은 하나의 parent를 가진다. 이 parent는 다른 쓰레드에 있을 수 있다. 대부분의 object의 parent는 socket이다. object의 parent의 parent가 socket인 경우도 있다. 따라서 socket을 시작으로 하는 tree가 생기게 된다. object는 자신의 children이 사라져야(shut down) 자신도 사라질 수 있기 때문에 이에 tree를 이용한다.
message passing에 관련이 있는 object와 관련이 없는 object로 해서 두 종류의 비동기 object가 있다.
message passing에 관련이 없는 object는 주로 connection management와 관련이 있다. 예를 들어 TCP listener object는 접속하려는 TCP 연결을 기다리고 각각의 새 연결에 engine/session object를 만든다. 비슷하게 TCP connector object는 TCP peer에의 연결을 시도하고 이것이 성공하면 연결을 관리하는 engine/session object를 만든다. 연결이 실패하면 다시 연결하기를 시도한다.
message passing에 관련이 있는 object는 data transfer를 다루는 object들이다. 이 object들은 두 부분으로 구성되어 있다. session object는 ZeroMQ socket과 통신을 하고 engine object는 네트워크 부분과 통신을 한다. session object는 한 종류만 있지만 engine object는 여러 type(TCP engines, IPC engines, PGM engines등)이 있다. 추가도 가능하게 되어 있다.
session은 socket과 message를 교환한다. message를 전달하는 것은 양방향으로 되어 있고 각각은 pipe object에 의해 다루어진다. 각각의 pipe는 쓰레드들 사이에 메시지를 빠르게 전달하는데 최적화된 lock-free 큐이다.
마지막으로, 모든 socket과 모든 비동기 object들에 의해 사용이 가능하고 global state를 저장하고 있는 context object가 있다.

8. Concurrency Model

멀티코어를 잘 이용하는 것은 ZeroMQ의 요건중의 하나다. 다시 말하면 코어가 증가하면 throughput이 증가하는 것을 의미한다.
전통적인 방법(critical sections, semaphores등)으로 여러 쓰레드를 사용하는 것은 성능을 향상시키지

않는다. 여러 쓰레드를 사용하면 코어가 여러개라도 하나의 쓰레드를 사용하는 것보다 느릴 수 있다.
ZeroMQ는 actor model이라 불리는 다른 모델을 사용한다. 쓰레드들 간의 통신에 쓰레드들 사이에 전달하는 비동기 메시지를 사용하는 것이다.
한 코어당 하나의 worker thread만을 사용한다. TCP engine 같은 각각의 내부 ZeroMQ object들은 특정한 worker thread에 연결되어 있다. 따라서 critical section이나 mutex나 semaphore같은 것을 사용할 필요가 없게 된다. 또한 이 object들은 CPU 코어 사이를 이동할 필요가 없게 되기 때문에 cache pollution을 피할 수 있게 된다.
이 방법은 기존의 멀티 쓰레딩의 문제를 사라지게 한다. 하지만 여전히 많은 object들 사이에 worker thread를 공유할 필요가 있다. 이벤트의 임의의 시퀀스를 다루어야 하고 object가 CPU를 오래 사용하면 안되도록 해야 하기 때문에 하나의 이벤트 루프를 사용하기 보다는 event-driven 방식으로 object를 관리하는 스케줄러가 필요하다는 것을 의미한다.
전체 시스템은 완전히 비동기적이어야 한다. blocking을 사용하는 object가 없어야 한다. blocking을 어떤 하나의 object가 사용하면 같은 worker thread를 공유하는 다른 쓰레드가 block될 수 있기 때문이다.
모든 object는 명백히든 내부적으로든 state machine이어야 한다. 동시에 실행중인 수백, 수천개의 state machine사이에 가능한 모든 통신을 다룰 수 있어야 한다. 특히 중요한것이 shutdown process이다.
완전히 비동기적인 시스템에서의 shutdown은 매우 복잡한 작업이다. 이 부분이 ZeroMQ에서 가장 복잡한 부분이다.버그 트래커를 보면 30-50%정도가 shutdown에 관련되어 있다.

배운점: 진짜로 performance와 scalibility를 고려한다면 actor model를 고려해보라.

9. Lock-Free Algorithms

lock-free algorithm은 커널이 제공하는 mutex나 semaphore를 사용하지 않고 쓰레드간에 통신을 하기 위한 방법이다. atomic compare-and-swap(CAS)같은 CPU의 기능을 사용해서 동기화를 한다. 이것은 하드웨어 레벨에서 lock이 이루어지기 때문에 진짜로 lock-free이지는 않다.
ZeroMQ는 pipe object에서 lock-free queue를 사용해서 user thread와 worker thread사이의 메시지를 전달한다. ZeroMQ가 lock-free queue를 사용하는 두가지 흥미로운 측면이 있다.
첫째로, 각각의 queue는 하나의 write thread와 하나의 reader thread를 가진다. 1:N 통신이 필요하면 여러 queue가 만들어진다. 이렇게 하면 writer들이나 reader들에의 동기화를 신경쓸 필요가 없어지기 때문에 매우 효율적이 된다.
둘째로, lock-free algorithm이 효율적이기는 하지만 여전히 우리가 원하는 만큼의 성능을 주지는 않았다.
이것을 향상시키기 위해 사용한 방법이 batching이다. 10개의 메시지가 있다고 하자. 이 메시지들을 하나씩 queue에 넣기 보다는 10개를 모은 후 한번에 queue에 저장하는 것이 낫다.
같은 방법을 reader쪽에도 적용할 수 있다. 메시지를 하나씩 읽기보다는 이 메시지들을 reader thread만이 읽을 수있는 queue의 특정영역(pre-read buffer)에 두고 reader thread가 한번에 읽을 수 있게 한다.

배운점: lock-free algorithm은 사용하기도 어렵고, 구현하기에도 문제 많고 디버깅하기도 좋지 않다. 가능하다면 기존의 가능한 방법을 사용하자. 너무 lock-free algorithm에 의존하지 말아라.

10. API

user interface는 제품에서 매우 중요한 부분이다. 라이브러리에서는 API가 될 것이다.
ZeroMQ의 초기 API는 exchange와 queue에 AMQP의 model을 따라 만들었다. 이것을 다시 처음부터 만들 때 BSD Socket API를 따라 만들었다. 이전에는 전문가가 사용하는 제품이었다면 이후부터는 누구든지 쉽게 사용할 수 있는 제품이 되었다. 커뮤니티는 10배로 커졌고 20개 이상의 언어의 바인딩이 만들어졌다.

배운점: 만들고자 하는 것을 알고 그에 맞는 UI를 만들어라.

BSD Socket API가 오래되었지만 많은 사람들이 알고 있고 쓰고 있는 것이기 때문에 ZeroMQ의 API를 배우기가 쉬워졌다. 또한 TCP, UDP, file, pipe같은 기존의 것과 같은 API를 사용하기 때문에 이것들과 같은 데서 사용될 수도 있고 기존의 것에 ZeroMQ를 쉽게 추가할 수도 있다. 그리고 BSD Socket API가 오래 사용되었다는 것은 이것이 좋은 디자인으로 만들어졌다고 말할 수 있는 것이다. 이것을 따른다는 것은 우리의 API도 좋은 디자인을 따라간다고 할 수 있다.

배운점: 제품을 만들 때 관련 제품을 살펴보라. 어떤 제품이 실패했고 어떤 제품이 성공했는지를 보고 배워라. 배울 수 있는 모든 것(idea, API, framework등)은 다 배워라.

11. Messaging Patterns

메시징 시스템에서 가장 중요한 문제는 사용자가 어떤 메시지를 어디로 보낼 것인지를 정하는 방법을 어떻게 사용자에게 제공할 것인가 하는 것이다. 이에는 두가지 접근법이 있다.
첫째로는 "do one thing and do it well"이라는 유닉스 철학을 따르는 것이다. 이것이 의미하는 바는 문제를 제한된 영역으로 제한하여 해결해야 하는 문제를 해결하는 것이다. 이것의 예로는 MQTT가 있다.
두번째로는 일반성에 초점을 맞추어 강력하고 구성이 가능한 시스템을 만드는 것이다. AMQP가 이것의 예이다. 이 모델은 어떤 routing algorithm도 정의할 수 있게 해주는 수단을 사용자에게 제공한다.
ZeroMQ는 첫번째 모델을 따른다. 첫번째 모델은 누구나 쉽게 사용할 수 있지만 두번째는 전문가정도는 되어야 사용이 가능한다.
특정 문제에 맞는 해결을 하는 것이 일반적인 해결을 하는 것보다 낫기는 하지만 그래도 사용자에게 다양한 기능을 제공하는 것이 좋다. 이 모순을 어떻게 해결할 수 있을까?
아래의 두단계로 해결할 수 있다.
1. 특정한 문제 영역을 다루기 위한 레이어를 정의한다.(transport, routing, presentation등)
2. 서로 관련이 없는 각 레이어의 구현을 만든다.
인터넷 스택에서 transport layer를 예로 들어보자. IP layer위에 다양한 서비스(TCP, UDP, SCTP등)가 존재한다. 이들은 서로 연관을 맺지 않는다. 따라서 쉽게 새로운 서비스를 추가할 수 있고 기존의 것을 제거할 수도 있다.
같은 원리가 messaging pattern에도 적용된다. messaging pattern은 transport layer위로 scalibility layer라 불리는 layer를 구성한다. 각각의 messaging pattern은 이 레이어의 구현이다.
publish/subscribe나 request/reply는 서로 연관없이 동작한다.

배운점: 복잡하고 다양한 측면을 가지는 문제를 해결하고자 할 때 하나의 일반적인 해결책이 최선이 아닐 수 있다. 대신에 문제 영역을 잘 추상화하고 각각의 레이어에 다양한 구현을 제공한다.

12. Conclusion

이 문서는 시스템적인 측면에서 large-scale distributed system을 만들면서의 우리의 경험을 이야기한다.