시스템프로그래밍

HW2: Concurrent Server

성능분석

20160051 박정환

I. 개요

주식을 사고 팔 수 있는 주식 서버를 두 가지방법으로 구현하였다. 한 가지 방법은 하나의 프로세스에서 각 클라이언트 별로 이벤트가 발생할 때마다 그것을 처리하는 이벤트 기반 서버이다. 다른 하나는 쓰레드 풀을 관리하여 클라이언트가 연결될 때마다 쓰레드를 할당하여 각각의 쓰레드에서 각각의 클라이언트의 요청을 수행하는 쓰레드 기반 서버이다.

본 명세서에서는 위에서 명기한 이벤트 기반 서버와 쓰레드 기반 서버의 성능을 다각도로 분석하였다. 먼저 동시 클라이언트의 수의 변화에 따른 각 서버의 처리율의 변화를 추적하였다. 두 번째 분석에서는 워크로드에 따른 쓰레드 기반 서버의 동시 처리율을 분석하였다. 우리 쓰레드 서버는 'Readers-writers' 문제를 따른다. 이 문제에서는 노드 당 Readers의 수에는 제한이 없지만 오직 하나의 Writer만 허락한다. 따라서 읽기 (주식 정보 보여주기) 단일 명령과 쓰기(주식 사고 팔기) 단일 명령에 따라 동시처리율에 차이가 있을 것이라는 가설을 세우고 실험을 진행하였다. 마지막 분석에서는 쓰레드 기반 서버에서 쓰레드 수에 따른 동시처리율의 변화를 분석하였다.

II. 성능 분석

동시처리율이란 단위시간(s)당 서버가 처리하는 클라이언트 요청의 수를 의미한다.

$$throughput = \frac{\#clients \times \#OrdersPerClient}{processing\ time(s)}$$

1. 확장성

동시 클라이언트 수의 변화에 따른 각 서버 방식의 동시 처리율의 변화를 분석하였다. 실험에 앞서 다음과 같은 가설을 세울 수 있었다.

<가설>

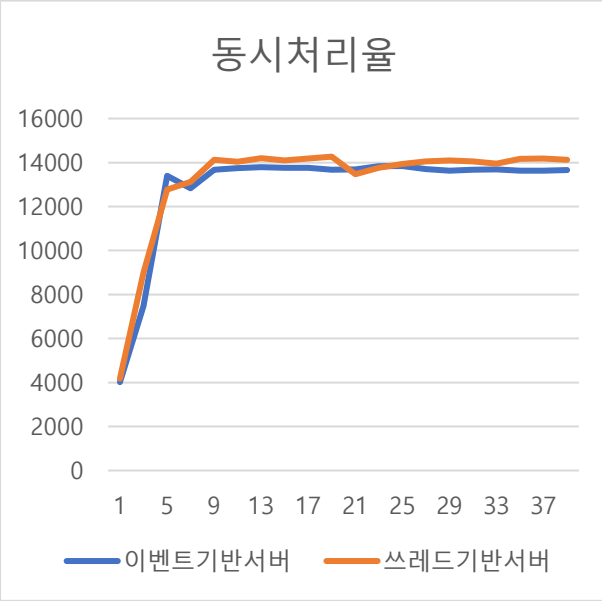
1. *클라이언트의 수가 증가함에 따라 과부하가 생겨 동시처리율이 떨어지는 지점이 발생할 것이다.*
2. *이벤트 기반 서버보다 쓰레드 기반 서버의 성능이 더 좋을 것이다.*

실험은 다음과 같은 정적 조건에서 진행되었다.

<실험1 조건>

- 쓰레드 수(Nthreads): 20

- 클라이언트당요청건수(OrderPerClient):100,000
- usleep, stdout 등 엄밀한 성능 분석을 방해하는 코드를 주석처리하였음.
- 클라이언트수(x) : 1<=x<=40
- 주식 종목 수: 10
- 실험환경: cspro



실험 결과, 다음과 같은 결과를 도출할 수 있었다.

<실험I 결과>

1. 이벤트 기반 서버와 쓰레드 기반 서버 간 성능의 유의미한 차이가 발생하지 않았다.
2. 동시처리율은 클라이언트 수가 증가함에 따라 로그적으로 증가하다가, 일정한 수준(14,000건 대)에 수렴하였다.
3. 클라이언트 수 초과에 따른 과부하 및 성능 저하는 발생하지 않았다.

결과1~3의 원인을 정확하게 진단하기는 어려우나, 각 결과와 다른 결과[1]가 도출되기 위해서는 클라이언트의 수가 실험에서 보다 훨씬 많아야 할 것으로 예상된다.

특히 결과3의 원인을 클라이언트의 수가 과부하가 생길 만큼 충분하지 않았기 때문이라고 판단하여 클라이언트의 수를 100까지 늘려서 실험을 진행해 보았으나 처리율은 여전히 13,000건 대로 정체된 것을 확인할 수 있었다.

2. 워크로드에 따른 성능 분석

실험II부터는 두 서버 중 쓰레드 기반 서버에 초점을 맞춰서 실험을 진행하였다. 실험II에서는 워크로드에 따른 동시처리율을 분석하였다.

이번 쓰레드 기반 주식 서버는 'readers-writers' 문제를 따른다. 이 문제 하에서 노드로 표현되는 주식 종목 하나당 readers의 수에는 제한이 없지만 오직 하나의 writer만 허락한다.

-

[1] 쓰레드 기반 서버가 이벤트 기반 서버보다 성능이 더 좋게 나오거나 클라이언트 수가 증가함에 따라 서버이 과부하가 발생하는 등의 결과를 말한다.

writer가 노드에 진입하면서 mutex lock을 걸기 때문이다. 따라서 다음과 같은 가설을 세울 수 있었다.

<실험II 가설>

1. *클라이언트의 요청이 read로 단일화되는 경우 write으로 단일화되는 경우보다 서버의 처리율이 높아질 것이다.*

. 실험II는 다음과 같은 조건 하에서 진행되었다. 실험II에서는 주식 종목의 수를 2로 제한하였는데, 이는 readers-writers 문제에서 발생하는 mutex lock 충돌의 문제를 자주 발생하게 하여 성능 차이를 확실하게 가시화할 수 있게 하기 위함이었다.

<실험II 조건>

- 쓰레드 수(Nthreads): 20
- 클라이언트당요청건수(OrderPerClient):100,000
- usleep, stdout 등 엄밀한 성능 분석을 방해하는 코드를 주석처리하였음.
- **주식 종목의 수를 2로 제한하였음.**

실험II의 결과는 다음과 같다.

<실험II 결과>

1. 워크로드 타입에 따른 유의미한 성능 차이는 발생하지 않았다.

결과가 가설대로 나오지 않아 꽤 실망스럽지만, 그 원인을 나름대로 분석해보면 다음과 같을 수 있다. read 함수는 재귀적으로 짜여 있고 모든 노드를 한번씩 순회한다. 따라서 한번 요청을 수행할 때마다 function call overhead가 발생한다. write(buy, sell)함수는 하나의 노드에 대해서만 작업을 수행하기 때문에, read 보다 기본적인 소요 시간이 작을 수밖에 없다.

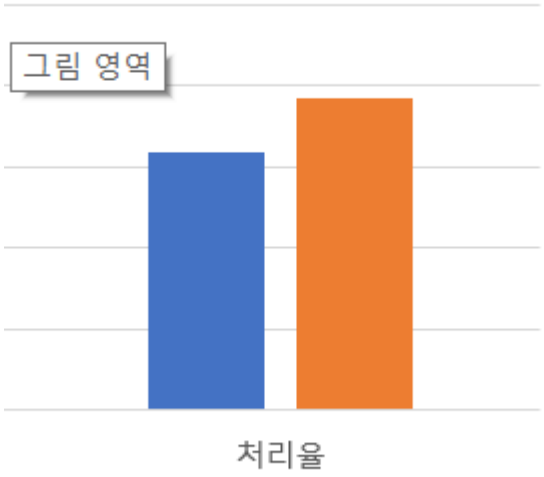
3. 쓰레드 수에 따른 성능 분석

쓰레드 기반 서버에서 쓰레드 수에 따른 동시처리율의 변화를 추적해보았다.

실험을 위해 한 번은 클라이언트 수가 쓰레드 풀에 있는 쓰레드 수 보다 적은 상황을, 다른 한 번은 쓰레드 수가 클라이언트 보다 많은 조건을 설정한 뒤 두 번의 실험을 진행하였다. 기타 조건은 다른 실험과 같다.

조건1: 클라이언트 수 10 / 쓰레드 수 5

조건2: 클라이언트 수 10 / 쓰레드 수 20



실험 결과는 예상한 대로 쓰레드 수가 많을 때 처리율이 좋다는 것이다. 쓰레드 풀 안의 쓰레드 수가 클라이언트의 수보다 적을 때는 요청이 처리되지 못하고 큐잉되는 클라이언트가 반드시 존재할 수밖에 없다. 반면 쓰레드 수가 클라이언트의 수보다 많을 때는 모든 클라이언트가 동시적으로 처리될 수 있기 때문에 Queuing Delay가 발생하지 않는다.

하지만 처리율이 쓰레드 수에 따라 선형적으로 증가하지는 않고 일정한 값에 수렴하는 구간이 발생한다. 그 원인이 무엇인지는 추후에 더 규명해야 할 것이다.