IOCP를 이용한 모델 기반 서버 성능 측정 도구 개발 방안

김규백
삼성전자 디지털솔루션센터
gyubaek.kim@samsung.com

A Method of Developing Model-based Server Performance Evaluation Tool using IOCP

G.B. Kim
Digital Solution Center of Samsung Electronics

요 약

대규모 클라이언트 연결과 다량의 네트워크 트래픽을 고속으로 처리해야 하는 서버의 성능을 정확히 측정하기 위해서는 네트워크 트래픽 부하를 발생시키는 것도 중요하지만, 실제 상황과 같은 서버의 오버헤드 요소를 모두 재현할 수 있어야 한다. 본 논문에서는 최근 멀티 채널 네트워크 I/O에서 뛰어난 성능을 보이는 Windows의 IOCP 기술을 이용해 실제와 같이 대량의 클라이언트를 생성해 서버와 동시에 세션을 연결하는 방법과 모델 기반으로 도출된 테스트 시나리오를 부하 발생 패턴에 적용하는 새로운 방안을 제시한다. 이를 활용해 향후 다양한 서버의 최대 성능을 보다 정확히 측정할 수 있다.

1. 서론

최근 온라인 게임과 같은 대규모 클라이언트의 네트워크 트래픽을 처리하기 위한 서버 네트워크 I/O 방식으로 Windows의 IOCP(I/O Completion Port)가 각광을 받고 있다[1]. IOCP는 서버의 처리 성능을 극대화 시켜 하드웨어 비용을 감소시키는데 크게 기여하고 있다[2]. 하지만, IOCP 기술이 멀티 채널 네트워크 I/O의 성능 향상에 폭 넓게 사용될 수 있음에도 불구하고 현재까지는 주로 서버 구현에만 활용되고 있다. 본 논문에서는 IOCP의 높은 효율성을 서버의 성능을 측정하기 위한 클라이언트 도구 개발에 활용하는 방안을 소개한다. 서버 성능 측정 도구를 IOCP를 이용해 클라이언트에서 구현할 경우 많은 수의 클라이언트 세션 연결과 트래픽을 발생시킬 수 있어, 실제에 가까운 테스트 환경을 통해 보다 정확한 측정 결과를 얻을 수 있다. 정확한 성능 데이터는 하드웨어/소프트웨어 최대 용량 계획(Capacity Planning)에 기준으로 적용되어 적절한 플랫폼 선정에 중요한 역할을 한다. 이러한 성능 측정이 현장에 서버가 설치, 운영되기 전에 도출될 수 있다면 매우 유용하다. 본 논문에서는 실제 트래픽의 종류와 구성이 명확하지 않은 초기 상태에서 성능 측정 결과를 얻는 것을 목표로 하고 있다. 이를 위해 [3]의 연구 결과와 같이 모델 기반의 예측 테스트 시나리오를 활용한다. 모델 기반 성능 평가 결과는 기본적으로 예측된 시나리오에 종속적이지만 시뮬레이션과 단순 예측에 의존하던 기존의 방법에 비해 매우 향상된 것이다.

본 논문의 구성은 다음과 같다. 제 2장에서는 기술 개요를 살펴보고, 제 3장에서는 성능 테스트 클라이언트 구현 방안을 도출하고, 제 4장에서는 IOCP 도입에 따른 효과를 분석하고, 제 5장에서 결론을 맺는다.

2. 기술 개요

2.1. IOCP 개요

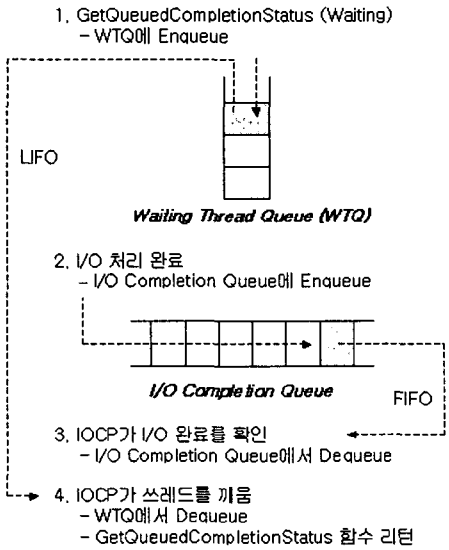
클라이언트 세션마다 연결을 유지해야 하는 서버의 부담을 줄이기 위해 기존에는 UDP가 많이 사용됐지만 클라이언트와의 상호작용이 많아지면서 서버쪽에서의 Push와 같은 기능을 구현하기 어렵고, 패킷 소실과 이에 따른 재전송 부담이 있기 때문에 최근에는 각광을 받지 못하고 있다. 또한, 세션마다 연결을 유지하는 TCP 방식에서도 각각의 클라이언트와 연결되어 네트워크 I/O를 처리하는 서버측 작업 쓰레드가 과다할 경우 Context Switching 오버헤드가 발생해 높은 성능을 기대하기가 어려웠다. 이런 기존의 문제점을 IOCP는 아래와 같은 특징을 통해 해결하였다.

IOCP는 [그림 1]과 같이 쓰레드간 동기화 객체 역할을 효율적으로 수행한다. 작업 쓰레드의 개수를 CPU 개수와 같이 유지하고 I/O 완료 시에 IOCP가 LIFO(Last In First Out) 방식으로 가장 최근에 대기 중인 작업 쓰레드에게 스케쥴링하기 때문에 Context Switching이 발생하지 않는다. 또한, 블로킹을 줄이기 위해 OS의 소켓 버퍼를 거치지 않고 사용자 주소 공간의 버퍼를 바로 이용해 I/O를 한다. 이러한 IOCP는 기존 소켓 관련 API들이 읽기/쓰기 과정에서 블로킹이 발생하는 동기 I/O 방식인 것과 달리 기본적으로 Overlapped라고 하는 비동기 I/O 방식을 함께 사용해 처리 속도를 높인다[4].

2.2. 부하 발생 도구

많은 부하가 발생했을 경우에도 올바르게 동작하는지를 검증하기 위한 수단으로 Rational의 TeamTest, Mercury Interactive의 WinRunner 등의 상용 부하 발생 도구가 많이 이용된다[5]. 이런 도구들은 실제 부하의 양을 조절하면서 Acceptance 테스트를 수행하지만, 최대 성능을 측정하기 보다는 Stress 테스트의 용도로 활용된다. 이 방법은 서버 쪽과의 연결 수가 적고, 트래픽의 대부분이 데이터베이스 작업과 같이 어느 정도의 작업 시간을 소요하는 시나리오를 포함하는 서버의 동작 부하를 발생시키는 데에는 유용할 수 있다[2]. 하지만 실제 대량의 연결을 하지 않고 적은 수의 연결에서 부하만을 발생시키기 때문에[6],

서버 측면의 가장 큰 오버헤드 요소를 포함하지 않아 최대 성능 측정이 정확할 수 없는 단점이 있다. 따라서, 단순히 부하 발생만으로는 서버의 최대 성능을 정확히 측정할 수 없다.



[그림 1] IOCP 동작

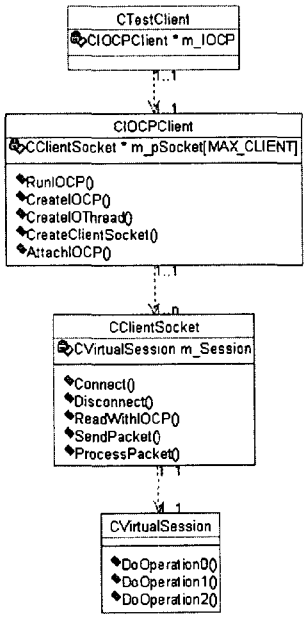
3. 구현 방안

3.1. 성능 평가 기준

대량의 세션 연결과 부하 발생 상황에서 서버의 최대 성능에 대한 판단은 임의의 클라이언트 세션에서의 반응 시간에 따라 이루어진다. 세션 시나리오를 구성하기 위해 모델 기반 테스트 방법 [3]이 이용된다. 또한, 반응 시간 기준을 설정하기 위해 SLA(Service Level Agreement) 개념을 도입해 사용한다. 만족스러운 성능은 주관적인 요소이기 때문에 SLA에 반응시간과 같은 성능 관련 품질 요소를 명시하는 방법은 바람직하다[7]. 본 논문에서 제시하는 방법은 [6]의 구분에 따르면, 서버의 최대 성능 결과를 파악하기 위해 클라이언트 동작을 판단 기준으로 하는 쪽으로 분류될 수 있고, 단순 패킷 수집과 같은 수동적 방식이 아니라 일련의 네트워크 트래픽을 발생시키는 능동적 성능 평가 방법에 속한다고 할 수 있다.

3.2. 소프트웨어 구조

이러한 IOCP를 구현하기 위한 기본 클래스 구조와 주요 멤버 함수 및 변수는 [그림 2]와 같이 정리된다. 최상위 CTestClient 객체에서 m_IOCP 멤버의 RunIOCP 메소드를 이용해 성능 테스트 클라이언트 생성을 최초로 시작한다. CreateIOCP 메소드를 이용해 IOCP 핸들을 생성하고(CreateIoCompletionPort 함수 이용), CreateIOThread 메소드를 통해 작업 쓰레드를 생성한다. 이때 작업 쓰레드의 개수는 CPU 개수와 동일하다. 작업 쓰레드가 시작되면 GetQueuedCompletionStatus 함수를 호출하며 IOCP 동작을 위해 대기하게 된다. 그리고 클라이언트 소켓을 생성하기 위해 CreateClientSocket 메소드를 호출한다. 최대 MAX_CLIENT 수만큼 일정한 시간 간격을 두고 루프를 돌며 소켓 생성 동작을 반복한다. 이 과정에서는 CClientSocket 클래스에서 명시해 놓은 메소드들이 사용된다. 블로킹을 없애는 비동기 방식인 Overlapped I/O를 사용하기 위해 소켓 기본 함수로 ReadFile, WriteFile을 내부적으로 사용한다. 먼저 Connect를 통해 연결이 이루어지고 난 뒤, CIOCPClient의 AttachIOCP를 호출해 IOCP 객체에 테스트 클라이언트 소켓을 연결시킨다(CreateIoCompletionPort 함수 이용). 여기까지의 과정이 수행되어야 IOCP가 동작하기 시작한다. 그리고 서버와 정해진 세션 시나리오에 따라 테스트 클라이언트가 자동으로 응답하기 시작한다. 이를 위해 CVirtualSession 클래스의 내용에 세부 동작이 구현되어 있다. ReadWithIOCP 메소드를 통해 서버로부터 메시지를 수신하고, 프로토콜에 따라 메시지를 해석한 후 ProcessPacket을 호출한다. ProcessPacket은 m_Session 객체를 통해 DoOperationX를 호출한다. CVirtualSession의 내용은 모델 기반의 테스트 시나리오로 정의되어야 한다. (여기서 사용된 MAX_CLIENT 개수가 곧 서버의 최대 성능을 나타내는 수치로서 이를 도출하는 방안은 4장에서 상세히 설명한다.)



[그림 2] IOCP 클래스 다이어그램

4. 효과

분석을 위해서 다음과 같이 모델을 설계한다.

[가정 1] 다중 쓰레드 동작 환경에서는 Context Switching 오버헤드가 가장 큰 성능 저해 요소이다[8].

[가정 2] 이더넷 네트워크 트래픽은 Self-Similarity 속성을 가진다[9, 10].

[가정 3] IOCP의 작업 스케줄링은 공정하다[4].

[가정 4] 메모리 사용 오버헤드가 없고 데이터베이스 처리와 같이 작업 시간을 요하는 시나리오를 포함하지 않는 클라이언트 네트워크 처리에 IOCP를 사용하면 Context Switching 오버헤드가 없어진다[2, 4, 8].

[가정 5] 대부분의 네트워크 처리 서버에서 사용하는 PDU(Packet Data Unit)는 크지 않다.

그리고, 아래와 같이 먼저 정의한다.
C (Capacity) = X packets/sec
S (Session) = {P_0, P_1, ···, P_N}
I (Interval) = Y sec
I를 간격으로 테스트 클라이언트 세션 S들이 차례대로 시작된다. 각각의 세션들은 P_N 다음에 P_0 패킷을 발생시키며 반복한다. 단일 세션에서의 패킷 구성은 서버가 처리해야 하는 전체 패킷 구성을 반영하고 있다. [가정 2]를 적용하면, 이와 같은 트래픽 발생 방법은 서버의 실제 트래픽에 가까운 상황을 재현하는데 효과적이다. 여기에서 테스트 클라이언트 생성 사이에 I 간격을 두는 이유는 네트워크 연결 설정에 기본적으로 소요되는 시간이 결과에 영향을 미치지 않도록 하기 위해서 이다. [가정 3]에 따라 패킷이 공정한 스케쥴링에 의해 처리되기 때문에 t 만큼의 간격이 지난 시점에서 동시에 처리해야 하는 패킷의 리스트는 IOCP를 통해 처리해야 하는 부하로서 $L(t) = \sum_{i=0}^{t} Pj$ (j = i % N) 이다. 그리고, 패킷을 처리하면서 발생하는 오버헤드를 O라고 했을 때, L(t)를 처리하기 위한 시간 T(t)는 t * (1/C + O)와 같다. [가정 4]에 의해 IOCP의 멀티 채널 네트워크 I/O의 효율성을 통해 O 요소가 0에 가깝게 근사된다고 할 때, $T(t) = \frac{t}{C}$ 가 된다. 이렇게 IOCP를 이용해 O 효과가 거의 없어지는 것은 [가정 1]에 따르면 엄청난 성능 향상 요인이 된다. 여러 클라이언트 세션이 동시에 활성화되어 있는 상황에서 하나의 세션을 구성하는 패킷들을 모두 처리하는데 걸리는 시간들의 합은 N * T(t)가 된다. 이 값이 사용자가 수용할 수 있는 수준 안에 들어있을 때 비로서 실제 환경에서와 같은 테스트 클라이언트 세션이 생성되었다고 할 수 있다. 여기에서 사용자가 수용할 수 있는 수준을 설정하기 위해 SLA(Service Level Agreement)를 도입한다. 따라서, $N * \frac{t}{C}$ <= SLA를 만족하는 최대 t를 구하면, 이 값이 클라이언트에서 생성할 수 있는 최대 테스트 클라이언트 수이다. 여기서 N은 세션 구성 시나리오 개수이고, C는 기본 네트워크 I/O 성능으로 특정 플랫폼에서 고정적인 요인이다. 따라서, 서비스 품질 측정을 위한 SLA 정의가 명확하면 서버의 최대 성능(= 최대 발생 가능 테스트 클라이언트 수)를 구할 수 있다. 최대 테스트 클라이언트 수는 점진적으로 증가시키면서 측정 가능한 값으로 이 과정을 통해 서버의 최대 성능뿐만 아니라 성능 튜닝 효과도 얻을 수 있다. 이 분석에 따르면 한 머신에서 발생시킬 수 있는 클라이언트 세션의 수준은 기존에 비해 놀랍게 향상이 된다. O에 대해 가정한 부분이 최대 성능 결과에 오차로 작용할 수 있으나, 그럼에도 불구하고 제안된 방법으로 시스템 개발 초기에 어느 정도 신뢰할 수 있는 데이터를 얻을 수 있다.

5. 결론

본 논문에서는 네트워크 I/O에서 뛰어난 성능을 나타내는 IOCP 기술을 서버의 최대 성능 측정을 위한 클라이언트 도구 개발에 활용하는 방법을 살펴 보았다. 이를 통해 실제 환경과 같은 대량 트래픽을 발생시킬 수 있고, 기존 방법과 시스템에서 재현하지 못했던 실제 다중 동시 세션 연결도 가능하게 하는 새로운 서버 성능 측정 방안을 제시하였다. 이 기술을 이용하고 적절한 테스트 시나리오 모델을 초기에 정의하면 통신망 서버, 웹 서버, 온라인 게임 서버, 홈 네트워크 서버 등 매우 다양한 서버의 최대 성능을 보다 정확히 측정할 수 있다.

향후 본 논문에서 사용된 기본 가정들에 대해 보다 다양하고 실증적인 분석을 수행하고, 많은 사례 연구를 발굴해 제안된 방안의 효용을 계속해서 검증할 계획이다.

6. 참고 문헌

[1] 김상근, " 온라인게임 플랫폼의 두 흐름" , 전자신문 테마 특강, 2004년 8월 17일.

[2] Gyu-baek Kim, " An Effective Processing Server for Various Database Operations of Large-scale On-line Games" , IASTED International Conference on Information and Knowledge Sharing(IKS) 2003, Arizona, U.S.A, November 2003.

[3] C.U. Smith and M. Woodside, " Performance validation at early stages of software development" , System Performance Evaluation: Methodologies and Applications. CRC Press, 1999.

[4] J. Richter, J. D. Clark, " Programming Server-Side Applications for Windows 2000" , Microsoft Press, 2000.

[5] K. Havelund, S. Stoller, and S. Ur., " Benchmark and Framework for Encouraging Research on Multi-Threaded Testing Tools" , PADTAD'03, Nice, France, April 22-26, 2003.

[6] Paul Barford and Mark E. Crovella, " Measuring Web performance in the wide area" , Performance Evaluation Review, August 1999.

[7] A. Sahai, et al., " Automated SLA Monitoring for Web Services" , IEEE/IFIP DSOM 2002, Montreal, Canada, Oct. 2002.

[8] A. Osterling, T. Benner, and R. Ernst, " Code Generation and Context Switching for Static Scheduling of Mixed Control and Data Oriented HW/SW Systems" , Proc. 4th Asia-Pacific Conference on Hardware Description Language, pp.131-135, Aug. 1997.

[9] M. Taqqu and V. Teverosky, " Is network traffic self-similar or multifractal?" , Fractals, Vol.5, No.1, pp.63-73, 1997.

[10] W. E. Leland, et al., " On the self-similar nature of Ethernet traffic" , IEEE/ACM Transactions on Networking, 2:1-15, 1994.