

경험한 이슈 중, 기억할만한 것들을 정리한 문서입니다.

1. CAS의 문제 (ABA Problem)

- **문제** : 락프리에서, 동일한 노드가 2번 Dequeue된다. ABA문제 발생
- **원인 분석** : 일반적인 CAS로는 노드의 유니크함이 보장되지 않음.
- **해결 방법**: Unique Count를 추가해 Double CAS 사용. 윈도우에서는 InterlockedCompareExchnage128이 Double CAS.

2. Double CAS 문제

- **문제** : Double CAS시 메모리 침범 문제 발생
 - **원인 분석** : 메모리 정렬로 인한 복합적인 문제 발생
- 1) 인터락 함수의 특징

인터락 함수의 특징

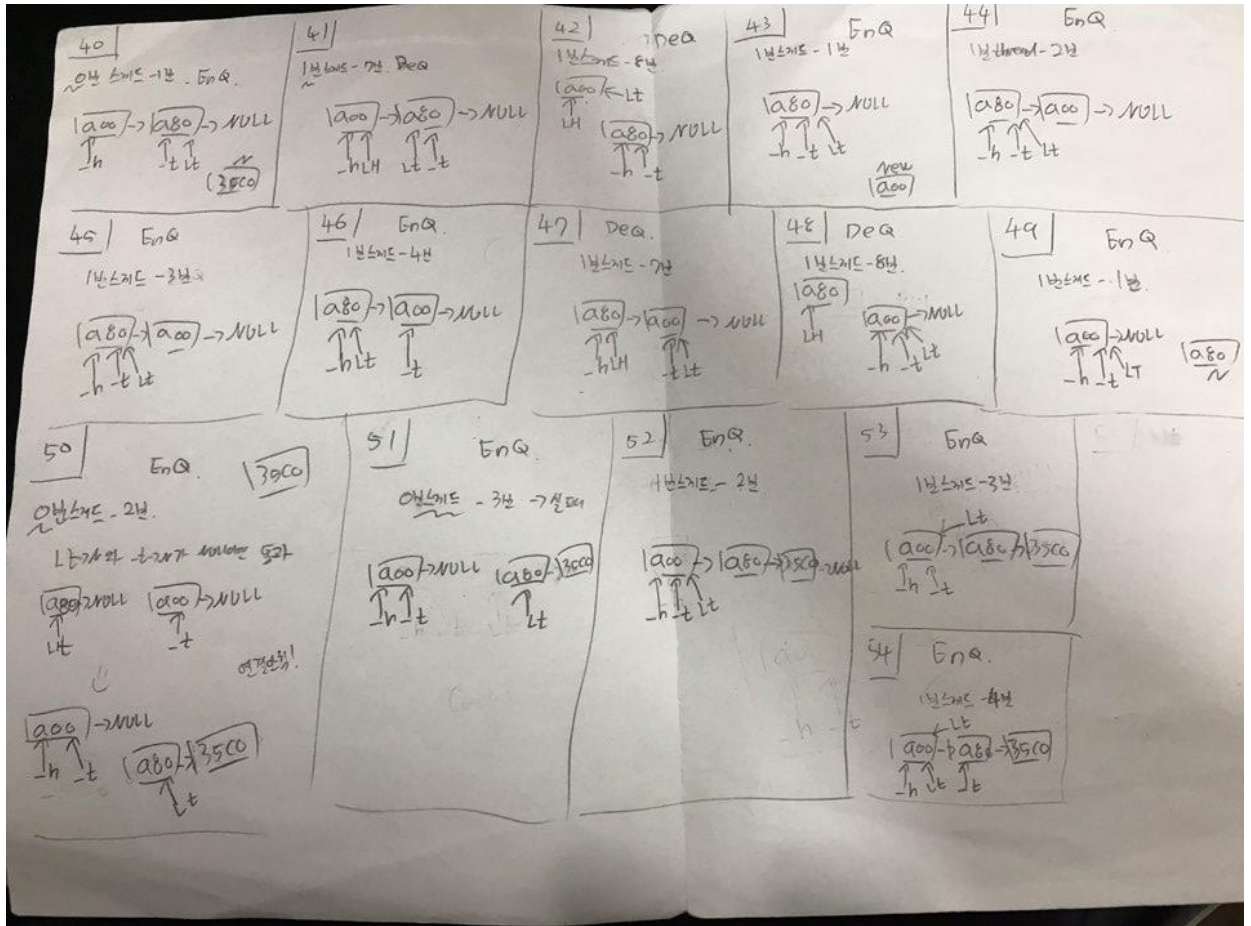
- 인자로 받은 변수가 존재하는, 실제 물리 메모리에 접근한다. (가상 메모리 아님!)
- 물론 MMU가 중간에 주소 변환을 해주는 것이다.
- 예를 들어, 인터락 32함수는 인자의 시작 주소부터 4바이트의 물리 메모리 영역에 접근한다.
- 인터락 64함수는 인자로부터 8바이트의 물리 메모리 영역에 접근한다.

- 2) 16바이트로 정렬된 메모리에 접근했을 때, 가상메모리 입장에서는 연속되어 있지만, 물리메모리 입장에서는 연속되지 않을 수도 있음.
- **해결 방법** : Alignas()함수로, double CAS에 들어가는 변수를 16바이트 정렬로 사용.

3. 락프리에서, Head의 Next가 null이 되는 상황

- 문제 : 락프리 큐의 Dequeue 함수에서 head의 next가 null이 되는 상황이 발생
- 원인 분석 : 인큐/디큐가 발생하는 모든 노드를 리스트에 보관. 크래시가 나는 순간 덤프파일로 리스트 확인.

1) 덤프를 추적한 결과



2) 발생 시나리오

- ◆ 0번 스레드가 Enq를 시도하는 중 컨텍스트 스위칭
 - ◆ 1번 스레드가 디큐->인큐->디큐 성공 후, Enq를 시도하는 중 컨텍스트 스위칭
 - ◆ 다시 0번 스레드가 Enq를 성공한 후 Tail을 이동시키지 못하면 Head의 Next는 Null이된다.
 - ◆ 이 때 0번 스레드가 디큐를 시도하면 Head의 next가 null이기 때문에 Crash가 발생한다.
- 해결 방법 : 정상적인 상황으로 판단하고, Dequeue 시, head의 next가 null이면 continue로 작업 반복.

4. 락프리와 SRWLOCK 성능 비교

- 이슈 : SRWLOCK과 락프리 중 어느 것이 더 빠를까?
- 분석 : SRWLOCK과 락프리 비교 결과, SRWLOCK이 월등히 빠르다.
 - 스레드 50개가 1천만번 InterlockedIncrement를 한 것과 SRWLOCK을 걸고 값을 ++.
 - 직접 제작한 프로파일링 사용

Profiling.txt - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

ThreadID	Name	Average	Min	Max	Call
8740	Interlock	64503397.700μs	64503397.700μs	64503397.700μs	2
8740	srwlock	10501700.200μs	10501700.200μs	10501700.200μs	2

1) 왜 SRWLOCK이 더 빠를까?

- 이것 저것 고민하고 자료를 찾다가 나온 결론은 **캐시라인과 캐시 무효화** 때문이다.

캐시라인

- RAM에서 캐시로 값을 가져오거나 값을 내보낼 때, 캐시라인 크기로 가져오고 보낸다. 이는 공간지역성을 최대한 활용하기 위함이다.
- 1바이트만 Read/Write해도 캐시라인 크기로 읽고 쓴다.
- 최근 Intel CPU의 L1/L2/L3 캐시라인은 64바이트이다.

캐시 무효화

- 캐시는 일관성이 보장되어야 한다. (cache coherency)
- 만약, 코어A/B의 캐시가 각각 int q의 값을 가지고 있는 상태에서, 코어 A가 q의 값을 변조할 경우, 코어 B에게 해당 값이 있는 캐시라인이 들어졌다는 내용이 전달된다. (MESI 프로토콜)
- 이후, 코어 B가 q에 접근하려고 할 때, Cache miss가 발생하며, 새로 데이터를 캐싱한다.

- 락프리는, CAS를 통과할 때 마다 캐시무효화 발생. → 매번 캐시라인 초기화 오버헤드
- SRWLOCK은 락을 풀때까지 캐시 무효화 발생하지 않음

2) 모든 락이 락프리보다 빠를까?

Profiling_크리티컬2.txt - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

ThreadID	Name	Average	Min	Max	Call
10360	Interlock	62786239.000μs	62786239.000μs	62786239.000μs	2
10360	Critical	77492227.400μs	77492227.400μs	77492227.400μs	2

- **결론1** : [SRWLOCK > 락프리 > Critical_Section] 순으로 속도가 빠르다. 물론 상황에 따라 다르다.
- **결론 2** : 락프리는 그냥 락이 없다는 것뿐 빠른 것은 아니라는 것.

5. SRWLOCK이 빠른 이유

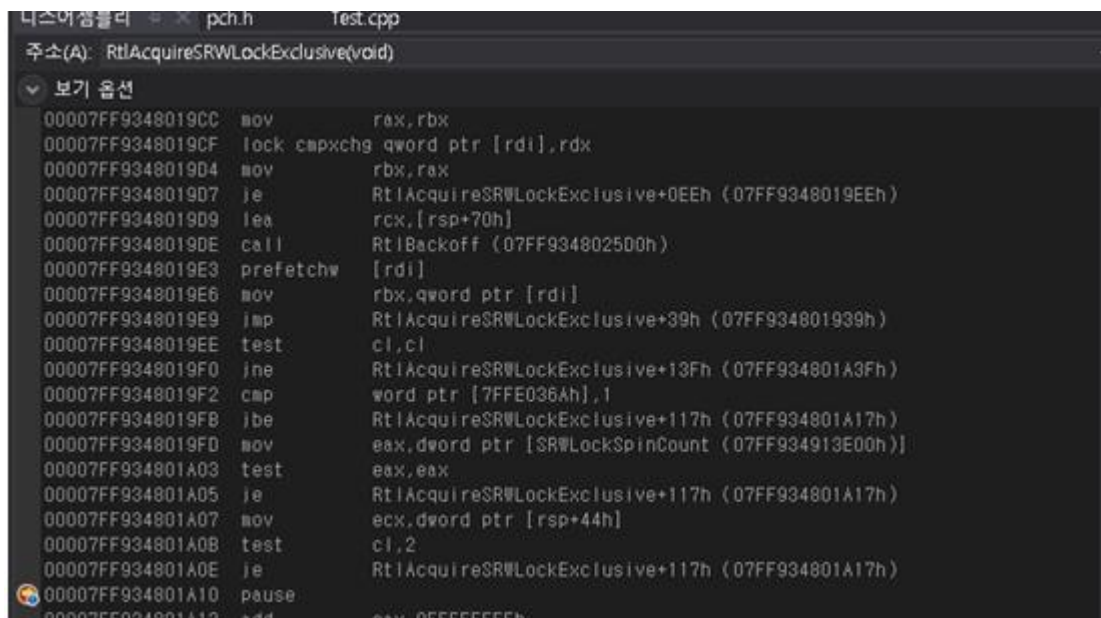
- 이슈 : SRWLOCK과 Critical_Section을 비교했을 때, SRWLOCK이 빨랐는데 그 이유는 무엇일까?
- 인터넷 검색 결과, Pause때문이라는 정보를 입수. 어셈으로 확인해보자.

Pause

- 하드웨어 영역의 어셈 명령어. OS는 전혀 관여하지 않는 명령어이다.
- 하이퍼 스레딩에서만 적용. 아니라면 NOP로 취급된다.
- 잠시 스레드를 쉬도록 해, 같은 코어에 접근하는 다른 스레드가 일을 더 많이 하도록 한다.
- 일정 횟수 루프를 돌면서 Pause를 실행한다.

1) Pause로 인해 락이 빨라질 수 있는 시나리오

- 하이퍼 스레딩에서 같은 코어를 사용중인 스레드 A와 B.
 - 스레드 A가 임계영역에 접근한 상태
 - 스레드 B가 같은 임계영역에 접근하길 시도
 - 스레드 B가 Pause를 하면서 쉬어준다면 스레드 A는 더 빨리 작업을 처리할 수 있음
 - 결과적으로 스레드 B가 임계영역에 들어가는 시간도 빨라질 것이다.
- 분석 1: SRWLOCK은 확실히 Pause 사용.



```
주소(A): RtlAcquireSRWLockExclusive(void)
보기 옵션
00007FF9348019CC mov     rax,rbx
00007FF9348019CF lock cpxchg qword ptr [rdi],rdx
00007FF9348019D4 mov     rbx,rax
00007FF9348019D7 je      RtlAcquireSRWLockExclusive+0EEh (07FF9348019EEh)
00007FF9348019D9 lea     rcx,[rsp+70h]
00007FF9348019DE call    RtlBackoff (07FF934802500h)
00007FF9348019E3 prefetchw [rdi]
00007FF9348019E6 mov     rbx,qword ptr [rdi]
00007FF9348019E9 jmp     RtlAcquireSRWLockExclusive+39h (07FF934801939h)
00007FF9348019EE test    cl,cl
00007FF9348019F0 jne     RtlAcquireSRWLockExclusive+13Fh (07FF934801A3Fh)
00007FF9348019F2 cmp     word ptr [7FFE036Ah],1
00007FF9348019F8 jbe     RtlAcquireSRWLockExclusive+117h (07FF934801A17h)
00007FF9348019FD mov     eax,dword ptr [SRWLockSpinCount (07FF934913E00h)]
00007FF934801A03 test    eax,eax
00007FF934801A05 je      RtlAcquireSRWLockExclusive+117h (07FF934801A17h)
00007FF934801A07 mov     ecx,dword ptr [rsp+44h]
00007FF934801A0B test    cl,2
00007FF934801A0E je      RtlAcquireSRWLockExclusive+117h (07FF934801A17h)
00007FF934801A10 pause
00007FF934801A12 add     eax,0FFFFFFFFh
```

- 분석 2: Critical_Section도 Pause 사용

디스어셈블리 - x pch.h Test.cpp

주소(A): RtlpEnterCriticalSectionContended(void)

보기 옵션

```

00007FF9347B981B mov     rdi,qword ptr [rbx+20h]
00007FF9347B981F mov     rax,rdi
00007FF9347B9822 and     edi,0FFFFFFh
00007FF9347B9828 and     eax,2000000h
00007FF9347B9820 test     rax,rax
00007FF9347B9830 setne   sil
00007FF9347B9834 xor     r10b,r10b
00007FF9347B9837 mov     al,1
00007FF9347B9839 xor     bpl,bpl
00007FF9347B983C mov     r14d,1
00007FF9347B9842 test     al,al
00007FF9347B9844 je      RtlpEnterCriticalSectionContended+8Fh (07FF9347B986Fh)
00007FF9347B9846 test     bpl,bpl
00007FF9347B9849 mov     r8d,3
00007FF9347B984F mov     rdx,rdi
00007FF9347B9852 cmov     r8d,r14d
00007FF9347B9856 test     rdi,rdi
00007FF9347B9859 je      RtlpEnterCriticalSectionContended+8Fh (07FF9347B986Fh)
00007FF9347B985B nop     dword ptr [rax+rax]
00007FF9347B9860 mov     eax,dword ptr [rbx+8]
00007FF9347B9863 test     r14b,al
00007FF9347B9866 jne     RtlpEnterCriticalSectionContended+0F7h (07FF9347B98D7h)
00007FF9347B9868 pause
00007FF9347B986A sub     rdx,r14

```

- 결론 1: SRWLOCK 이 Critical Section 보다 빠른 이유는 Pause 가 아니다. 과거에는 맞을지도 모르지만 현재는 아님.
- 결론 2: 동기화 객체 사이즈 작음/ 락 진입 시 비트값 변경이 끝/ Shared 모드 제공 등으로 SRWLOCK 이 빠른 것으로 추측된다.

6. ERROR_SEM_TIMEOUT

- 문제 : 더미 테스트 중, 서버가 더미를 끊는 상황이 발생.

- 원인 분석

1) 서버에서, 접속하는 모든 세션에 상태값(커넥트 상태, 로그인 상태 등..)을 부여한 후 서버가 끊는 유저에 대해 로그 확인.

➔ 다양한 상황에서 발생. 확인 불가

2) 네트워크 모듈 코드 확인. GQCS의 리턴값이 FALSE면 GetLastError로 확인.

➔ 121에러가 발생했다.

121에러 (ERROR_SEM_TIMEOUT)

- 네트워크 단절로 인해 발생하는 에러

<https://blogs.msdn.microsoft.com/oldnewthing/20140717-00/?p=483>

(MSDN 공식 답변. 요약하면 TCP에서 retransmission 횟수만큼 재전송을 실패하면 발생)

- 원인 분석 2

■ 테스트 환경 상, 더미가 있는 PC 와 서버가 있는 PC 는 스위치나 라우터를 거치지 않고 직접 연결되어있는데 과한 통신으로 인한 패킷 드랍으로 추측.

■ 클라가 서버에 연결한 후, 클라의 랜선을 강제로 뽑은 다음 서버가 클라로 패킷을 보내봤는데, WireShark 로 확인해보니 패킷 5 회 재전송 후 121 에러가 발생했다.

- 해결 방법

1) 재전송 횟수 수정 ➔ 실패

■ 윈도우에서 regedit 을 열고 아래 경로에 TcpMaxDataRetransmissions 를 추가하면 재전송 횟수 제어 가능.

■ 10 회로 늘린 후 테스트를 해봤는데, 여전히 발생했다.

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters

2) 네트워크 과부하 감소 ➔ 해결

■ 더미가 100 밀리세컨드에 1 회 패킷을 보내는 것을 1000 밀리세컨드마다 1 회씩 보내는 것으로 수정.

■ 서버가 더미를 끊는 상황 발생하지 않음.

7. 아파치가 재시작되는 문제

- 문제 : Dummy → 아파치 → PHP에서 DB에 UPDATE/SELEC를 하는 중 갑자기 Dummy가 종료되는 상황 발생.
- 원인 분석
 - 1) Dummy 로그 확인
 - HTTP 통신 중, 아파치가 먼저 접속을 끊어 Dummy가 Crash를 내면서 프로그램이 종료된 것
 - 2) 아파치 로그 확인
 - 정말 Dummy가 죽은 그 시간에 아파치가 재시작.
 - 오류 코드 c0000374(힙 침범)
 - 3) 이벤트 뷰어 확인
 - httpd.exe가 c0000374오류코드로 종료되었다고 남아있었다.
 - 4) 스택 오버플로우 확인
 - 비슷한 질문 확인. 답변에 PHP 공식 홈페이지 링크가 있었다.
 - 확인해보니 아파치 재시작 버그가 수정되었으니 PHP버전 업그레이드 하라는 내용
- 해결 방법 : PHP 버전을 7.2 스레드 세이프 버전으로 업그레이드 하니 깔끔하게 해결되었다

8. 아파치의 메모리가 계속 증가하는 문제

- **문제** : Dummy에서 아파치를 통해 PHP를 거쳐 DB에 UPDATE/SELECT 테스트 중, SEELCT / UPDATE TPS가 점점 떨어지기 시작.
- **원인 분석**
 - 1) **작업 관리자 확인**
 - 서버 PC의 작업관리자에 메모리가 90% 사용 중이었다.
 - 아파치가 거의 80%를 쓰고 있었다.
 - 메모리가 계속 늘어나, 페이지 폴트가 발생해 전체적으로 성능이 내려간 상황으로 추정
 - 2) **스택 오버플로우 확인**
 - 비슷한 질문을 찾았는데, http-mpm 설정파일의 MaxConnectionsPerChild를 건드리라는 것.
 - MaxConnectionsPerChild는 아파치에 연결 가능한 최대 수로, 이 이상 커넥트되면 아파치 프로세스를 kill했다가 다시 살린다.
 - 답변에 더 자세한 설명은 없었지만, 커넥트 할 때 마다 쌓이는 정보가 있는데 그게 메모리를 차지한 것으로 추측.
- **해결 방법** : MaxConnectionsPerChild를 100만으로 수정해, 100만명마다 프로세스를 강제로 kill. 문제가 해결되었다.

9. 포트 부족 문제

- 문제

- 더미테스트를 하는데, php에서 mysql connect나 fsockopen 시 에러가 발생했다.
- 에러 메시지는 다양한데, 결론은 어쨌든 실패했다는 것이다.

- 원인 분석

1) Netstat -a 확인

- 거의 모든 포트가 사용중이며, 대부분이 TIME_WAIT 상태.
- 더미가 과하게 접속을 시도함으로써 포트 고갈현상이 발생한 것.

- 해결 방법

1) 동적 포트 확장 → 실패

- 윈도우에서 동적포트 확인 : cmd에 netsh int ipv4 show dynamicportrange tcp
- 윈도우에서 동적포트 확장 : cmd에 netsh int ipv4 set dynamicportrange tcp start=32767 num=32768 store=persistent 숫자부분을 잘 조절하면 된다.
- 그래도 여전히 동일한 문제가 발생했다.

2) TIME_WAIT 시간 감소 → 성공

- 포트 수가 아무리 많아도 TIME_WAIT으로 포트가 남아있기 때문에 똑같이 포트 고갈현상이 발생한 것.
- TIME_WAIT 시간을 낮춘 후 테스트하니 정상적으로 작동했다.

10. MySQL이 CPU를 100% 점유하는 상황

- 문제 및 원인 파악 : 서버 컴퓨터가 버벅거리기 시작했다. 작업관리자를 확인해보니 MySQL이 CPU를 100% 점유한 상태였다.

- 시도한 해결 방법

1) Sync_binlog 설정 수정 → 성과 있음

- 해당 DB는 Replication되어 있었으며, 그 중 Master였음.
- 과한 바이너리 로그 전송으로 인해 MySQL이 CPU를 많이 사용하는 것이 아닐까 추측.
- sync_binlog 설정을 1에서 100으로 수정해봤다.
- 수정 후, SELECT / UPDATE의 쿼리 TPS는 올라갔지만 CPU는 여전히 100%

Sync_binlog

- Slave와의 동기화를 결정한다.
- 1이면 쿼리 1개가 발생할 때 마다 slave와 동기화 발생
- Replication에 사용되는 옵션이다.

2) innodb_flush_log_at_trx_commit 설정 수정 → 성과 있음

- MySQL 공식 홈페이지를 검색하다가 innodb_flush_log_at_trx_commit라는 설정을 찾았다.

innodb_flush_log_at_trx_commit

- 바이너리 로그를 디스크에 저장하는 시점을 설정. My.ini에서 설정 가능
- 0: 초당 1회씩 트랜잭션 로그 파일(innodb_log_file)에 기록 (1초마다 디스크 기록 및 커밋)
- 1: 트랜잭션 커밋 시 로그 파일과 데이터 파일에 기록 (새로 커밋될 때 마다 디스크에 기록 및 커밋)
- 2: 트랜잭션 커밋 시 로그 파일에만 기록, 매초 데이터 파일에 기록 (새로 커밋되면 즉시 커밋. 1초가 되면 디스크 기록)

- 1로 설정되어있었으며, 이는 UPDATE 마다 디스크와 I/O작업이 있었던 것.
- 수정 후, SELECT / UPDATE의 쿼리 TPS는 올라갔지만 CPU는 여전히 100%

3) 병목현상 확인을 위한 PHP 코드 프로파일링 → 성과 있음

- MySQL가 병목현상이 아닌 것 같아, 병목현상을 다시 찾기 위해 PHP 코드 프로파일링.
- MySQL에 Connect하고 Query를 전송하는 부분을 프로파일링.
- Query는 약 0.001초, Connect는 최대 0.1초가 걸리는 내용 확인.
- 병목현상은 PHP이며 그 중에서도 MySQL에 Connect할 때로 확인.

4) PHP 지속연결 (Persistent connect) 사용 → 성과 있음

- PHP의 Mysql connect 관련 내용을 찾아보는 중, PHP 공식 홈페이지에서 지속연결이라는 것을 찾았다.

PHP의 지속연결

- 보통, TCP 통신은 Connect/Disconnect의 부하가 심하다. (3way handshake/4way handshake)
- 지속연결은 연결을 유지해 Connect와 Disconnect가 발생하지 않게 하는 방법.
- mysqli_connect('p:', \$db_host, ...) 처럼 'p'를 붙이면 지속연결로 연결된다.

- 수정 후, SELECT / UPDATE의 쿼리 TPS는 올라갔지만 CPU는 여전히 100%

5) OPcache 사용 → 최종 해결

- Php.ini의 기능들을 검색하다가 OPcache 기능을 알게되었다.
- OPcache 기능 사용 후, MySQL의 CPU 점유율이 50%로 감소되었다.

PHP의 OPcache

- php는 기본적으로 컴파일 -> 실행 -> 출력의 절차를 거치는데, Opcache를 사용하면 컴파일된 코드를 메모리에 캐싱해둔다
- 즉, 컴파일을 1회만 실행하여 컴파일 오버헤드를 줄여줌

11. 네트워크 과부하

- 문제 : DB에 UPDATE 중, Lost connecting to MySQL server during query...(2013) 에러 발생
- 원인 분석
 - 1) 성능 모니터 확인
 - 네트워크 송수신 바이트가 계속 증가
 - 그러다 어느 순간 Lost connecting to MySQL server during query...(2013)에러가 발생
 - 2) 서버 별 송수신 바이트 확인
 - 배틀/채팅/매칭서버의 송수신 바이트 확인
 - 채팅서버가 가장 과하게 나타남 (더미가 과하게 채팅을 하는 상황이었음)
- 해결 방법
 - 1) 서버 위치 이동
 - 기존에는 1U 1개에 [배틀/채팅/매칭/아파치], 다른 1U에 [더미]가 있었다.
 - 채팅이 과하기 때문에 [배틀, 매칭 아파치], [더미, 채팅]으로 서버 위치를 이동시킴
 - 더 이상 2013에러가 발생하지 않음.