기한이 촉박해 보이고 내용을 내가 관여할 것은 아닙니다.

의견을 구한것이니 그대로 가감없이 의견을 전합니다.

정내훈 교수님께 참조로 같이 보냅니다.  
  
  
  
- 전체적인 평 -  
  
논문의 주제를 충분히 알아보지 않고 진행한 것이 아닌가 생각한다.  
아직 완성된 논문이 아니다.

기한내에 완성하기 어려우리라 생각한다.  
  
  
  
- 요약에서 주장하고 있는 것들 -  
  
C++ 스마트 포인터를 lock-free 스마트 포인터로 재구성한다    
-> 알겠다  
  
ABA가 발생하지 않음을 논증    
-> 논문에서 이것을 논증하는 부분을 찾지 못했다

> **정내훈 교수님 수정** -> ABA문제를 발생이 아님

>> **“LFCB 재사용으로 정확성을 가진다”로 수정**   
  
다른 스마트 포인터와 비교 성능 측정  
-> 벤치마크 프로그램을 사용하여 실험하였다  
  
멀티쓰레드에 최적화된 스마트 포인터임을 확인  
-> 왜 그렇다는 것인지 알 수 없었다  
  
MMORPG 서버의 성능을 향상시킬 수 있다  
-> 얼마나, 어떻게 했다는 것인지 찾을 수 없었다

> **정내훈 교수님 제거** -> 내용에 맞지 않다  
  
- 논문을 따라가며 살펴본 내용 -  
  
1장부터 전체    
  
주어와 서술어를 맞춰야 한다. 부드럽게 읽히도록 고쳐야한다.

> 더 필요함  
  
2장  
C++11부터 스마트 포인터를 제공  
**-> C++11에서는**    이유 - 이전의 auto\_ptr도 스마트포인터 임  
  
2장  
스마트 포인터는 포인터처럼 동작하는 클래스 템플릿으로, 참조하는 객체에 대한 수명을 가지며  
**-> 스마트 포인터는 포인터처럼 동작하는 클래스 템플릿으로, 참조하는  
    객체의 수명을 관리하며 수명이 다한 객체의  메모리를 자동으로 해제한다**  
  
   \* 이와 같이 표현을 다듬어야 할 부분이 많이 있습니다.

> 논문 수정이 더 필요함

2장(3쪽 시작부분)  
    shared\_ptr와 weak\_ptr는 멤버변수로 control\_block을 참조하는 ctr과 객체를  
    참조하는 ptr을 갖는다. 하지만 두 멤버변수의 원자적인 동시수정이 불가능하여  
    멀티쓰레드 환경에서 Data-Race를 초래하며, C++는 이러한 이유로 멀티쓰레드  
    환경에서 스마트 포인터의 안전한 동작을 보장하지 않는다[9].  
  
    -> 참고문헌 [9]에서 이런 표현을 찾을 수 없음  
     
내 해석으로는 data race를 방지하려면 이렇게 사용하라는 내용임.  
   
    이 부분이 잘못된 것이라면 이 논문은 시작부터 잘못된 것임.  
     
  
아래는 웹사이트 내용이고 C++17, C++20(draft)에서도 변동된 내용은 없음  
  
All member functions (including copy constructor and copy assignment) can be called by multiple threads on different instances of shared\_ptr without additional synchronization even if these instances are copies and share ownership of the same object. If multiple threads of execution access the same shared\_ptr without synchronization and any of those accesses uses a non-const member function of shared\_ptr then a data race will occur; the shared\_ptr overloads of atomic functions can be used to prevent the data race.  
  
// 이후 뒷부분  
  
To satisfy thread safety requirements, the reference counters are typically incremented using an equivalent of std::atomic::fetch\_add with std::memory\_order\_relaxed (decrementing requires stronger ordering to safely destroy the control block).

>> [9]에서는 멀티쓰레드에서 std::atomic을 이용해 shared\_ptr을 사용할 수 있다고 설명하고 있다.

>> 하지만, std::atomic을 이용하면 성능이 매우 느리다!

>>> 그래서 멀티쓰레드에서 높은 성능을 가진 Lock-Free를 구현하는 것이다.

>>>> 이 부분에 대한 설명을 **논문에 추가하자!**

3장  
  
내가 잘 알지 못하는 부분이라 자세히 읽지는 않았다  
그렇지만 이 정도로 자세하게 써야 하나 하는 의문이 들었다.  
너무 상세하고 장황해 보인다는 의미이다.  
**> 3장 내용을 간략하게**  
4장  
3장의 분량에 비해 너무 작아 제대로 비교평가를 한 것인가 의심이 든다.

**> 벤치마크 프로그램 내용 부족 or 3장이 너무 길다**  
  
4.1.1 atomic함수를 이용한 shared\_ptr  
Node가 std::shared\_ptr로 구현된 리스트는 ptr과 ctr에서 Data Race가 발생할 수 있다. 이를 방지하기 위해서 모든 std::shared\_ptr의 읽기, 쓰기작업은 atomic\_load()와 atomic\_store()를 이용하며, 이는 전역 lock을 사 용한다[11].  
  
-> std::shared\_ptr에서 data race가 발생하지 않도록 할 수 있음을 쓰고 있어  
**논문 2장에서 지적한 내용과 모순**된다.

**>> std::shared\_ptr 사용하는 방법을 언급하자**  
  
[Table 2]가 이 논문의 중요한 실험결과로 보이는데 설명이 너무 허술하고  
어떻게 더 낫다는지 이해하기 어렵다. ATSP의 -값의 의미가 궁금하며 숫자를  
구분한 리터럴이 잘못되어 있는데 어떤 의도인지도 궁금하다. (4'96'9107)  
  
**4장 실험부분을 더 잘 써야한다**  
  
5장  
벤치마크 프로그램을 통해 고성능 Lock-Free 스마트 포인터임을 확인하였다.  
-> 논문을 읽고 이것을 확인할 수 없었다.

결론.

>> 전문가도 이해하지 못하는 내용이 많고, 글을 이해하지 못하는 부분도 많다.

1. 작문을 좀더 신경써야함
2. Lock-free shared\_ptr이 왜 필요한지 내용 삽입.
3. 3장의 코드 내용이 번잡하다.
4. 4장의 벤치마크 프로그램 결과 내용이 부족하다.