분산 시스템에서 고장 감내성의 향상을 위한 적응형 체크포인팅 프로토콜

이용호, 장태무
동국대학교 컴퓨터공학과

Adaptive Checkpointing Protocol for Improving of Fault Tolerance in Distributed System

Yongho Lee, Taemu Chang
Dept. of Computer Engineering, Dongguk University

요 약

비동기 체크포인팅 프로토콜은 분산 시스템에서 고장 감내성을 제공하기 위한 방법 중 하나다. 이 방법은 모든 프로세스가 독립적으로 자신의 지역 체크포인트를 두고 어느 한 프로세스에서의 고장 발생 시 가장 최근의 체크포인트에서부터 롤백을 하는 것이다. 하지만 이 방법은 어느 한 프로세스에서의 고장 발생이 다른 프로세스의 롤백까지 유도하는 캐스캐이드 롤백을 발생시킬 수 있다는 단점이 있다. 본 논문에서는 고장 감내성의 수준을 높이기 위하여 비동기 체크포인팅 프로토콜을 사용하면서도 캐스캐이드 롤백을 막을 수 있는 적응형 체크포인팅 프로토콜을 사용한다. 또한, 본 논문에서는 저장 장치와 네트워크의 발달을 전제로 하여 기존의 방법과는 다른 접근을 한다. 프로세스사이에 오고가는 모든 메시지의 복사본이 서버쪽의 중재자를 통하여 서버에 있는 기계 상태 테이블에 저장된다. 이렇게 하여 서버에는 모든 지역 기계의 상태가 저장되어 기계 고장이 발생했을 경우에 고장이 발생한 기계의 복구에 사용된다.

1. 개 요

분산 시스템에서 고장 감내성(fault tolerance)을 제공하기 위한 방법으로는 체크포인팅 프로토콜(checkpointing protocol)[1][2][3][6]과 메시지 로깅 프로토콜(message logging protocol)[3][4]이 있다.

체크포인팅 프로토콜은 두 가지 방법으로 분류될 수 있다. 첫 번째는 동기 체크포인팅 프로토콜(synchronous checkpointing protocol)[3][6]인데 이것은 캐스캐이드 롤백(cascade rollback)을 피하기 위하여 프로세스가 안정 저장 장치에 일관된 전역 상태를 저장하는 것이다. 하지만 이 방법은 동기화 한 전역 체크포인트를 두기 위해 모든 프로세스가 정지되어야 한다는 것과 더 많은 통신이 필요하다는 단점이 있다. 두 번째는 비동기 체크포인팅 프로토콜(asynchronous checkpointing protocol)인데 이것은 모든 프로세스가 독립적으로 자신의 지역 체크포인트를 두는 것이다. 하지만 이 방법은 어느 한 프로세스의 고장 발생이 다른 프로세스의 롤백까지 유도하는 캐스캐이드 롤백을 발생시킬 수 있다는 단점이 있다.

메시지 로깅 프로토콜(message logging protocol)은 프로세스가 프로세스들 사이에 오고가는 메시지를 저장하고 있는 것이다. 그러다가 고장이 발생하면, 고장이 발생한 프로세스는 로깅 메시지를 이용하여 롤백을 한다. 이 방법은 동기 체크포인팅 프로토콜보다 통신 부하는 적지만 복구 시에 복잡도가 높아진다는 단점이 있다. 메시지 로깅 프로토콜은 로깅 방법에 따라 송신자 기반 메시지 로깅(sender-based message logging)[4]과 수신자 기반 메시지 로깅(receiver-based message logging)으로 분류될 수 있다.

최근에는 기술의 눈부신 발달로 인하여 저장 장치 부하와 네트워크 전송 부하는 예전과 비교하여 많이 줄었으며 앞으로는 더욱 줄어들 전망이다. 이런 환경이 된다면 지금까지의 고장 감내도 기법들에 많은 변화가 생기고 접근 방법도 달라질 것이다.

본 논문에서는 저장 장치와 네트워크의 발달을 전제로 하여 기존의 방법과는 다른 접근을 한다. 통신 부하를 줄이기 위하여 비동기 체크포인팅 프로토콜에 기반을 하여 각각의 프로세스가 자신의 지역 체크포인트를 독립적으로 두되 이 프로토콜의 단점인 캐스캐이드 롤백을 피하기 위하여 적응형 알고리즘[1][2]을 사용한다. 이것은 캐스캐이드 롤백을 유발하는 잠재적 고아 메시지(potential orphan message)[5]를 찾아내어 체크포인트를 추가하도록 하는데, 이런 방법을 적응형 체크포인팅 프로토콜이라 한다. 기존의 적응형 방법이나 혼합형 방법도 잠재적 고아 메시지를 찾아내어 캐스캐이드 롤백을 줄이는 방법인데, 차이점은 기존의 혼합형 방법과는 다르게 본 논문에서는 메시지 로깅을 사용하지 않고 체크포인트의 추가만으로 캐스캐이드 롤백을 피하려한 것이고, 기존의 적응형 방법과는 다르게 본 논

문에서는 모든 예외 상황을 검사하여 체크포인트를 추가하여 캐스캐이드 롤백을 피하는 것이다. 이렇게 되면 기존의 적응형 방법보다는 체크포인트의 수가 많아지겠지만 롤백을 하는 시간을 최소한으로 줄일 수 있어 전체 수행시간이 짧아진다. 또한 프로세스사이에 오고가는 모든 메시지의 복사본이 서버쪽의 중재자(coordinator)를 통하여 서버에 있는 기계 상태 테이블(machine state table)에 저장된다. 이렇게 하여 서버에는 모든 지역 기계의 상태가 저장되어 지역 기계 고장이 발생했을 경우에 고장이 발생한 기계의 복구에 사용된다. 이렇게 하여 프로세스의 고장뿐만 아니라 기계의 고장도 복구할 수 있게 된다.

본 논문의 구성은 제 2장에서 동기와 관련 연구를 고찰하고 제 3장에서는 시스템 모델을 설명한다. 제 4장에서는 구현과 성능 평가를 보이고 마지막으로 제 4장에서는 결론과 향후 연구 방향에 대해 기술한다.

2. 동기와 관련 연구

최근에는 기술의 눈부신 발달로 인하여 저장 장치 부하와 네트워크 전송 부하는 예전과 비교하여 많이 줄었으며 앞으로는 더욱 줄어들 전망이다. 이런 환경이 된다면 지금까지의 고장 감내성 기법들에 많은 변화가 생기고 접근 방법도 달라질 것이다.

체크포인팅과 롤백을 이용한 복구는 분산 시스템에서 고장을 복구하는 방법이다.[1] 하지만 이 방법은 어느 한 프로세스의 고장 발생이 다른 프로세스의 롤백까지 유도하는 캐스캐이드 롤백을 발생시킬 수 있다는 단점이 있다. 이전의 연구에서는 캐스캐이드 롤백을 피하기 위하여 몇 가지 방법을 사용하였다. 첫 번째는 캐스캐이드 롤백을 피하기 위한 간단한 가정을 초기에 두는 것이다. 두 번째는 캐스캐이드 롤백을 발생시키는 잠재적 고아 메시지가 나타나면 그 메시지를 저장해 두는 것이다.[5] 이런 방법을 메시지 로깅 프로토콜이라 한다. 마지막으로 캐스캐이드 롤백을 피하기 위하여 안정 저장 장치에 일관된 전역 상태를 저장해 두는 것이다.[6] 이런 방법을 동기 체크포인팅 프로토콜이라 한다.

비동기 체크포인팅 방법은 프로세스 사이의 동기화가 필요 없다는 장점이 있다. 본 논문은 비동기 체크포인팅 방법의 장점을 취하면서도, 이 방법의 단점인 캐스캐이드 롤백을 피하기 위하여 적응형 알고리즘을 함께 사용한다. 본 논문에서는 캐스캐이드 롤백을 발생시키는 잠재적 고아 메시지가 나타나면 서버의 중재자가 그 프로세스에게 체크포인트를 추가하도록 하는 방법을 사용한다.

그림 1은 어느 한 프로세스의 고장을 복구하기 위하여 초기 상태까지 롤백되는 캐스캐이드 롤백의 경우를 예로 든 것이다. 여기서 P는 프로세스를 가리키고, $C_{1,1}$은 프로세스 1의 체크포인트 1번을 가리킨다. P_2를 $C_{2,1}$까지 롤백하기 위해서는 메시지 m5가 완료되지 않았기 때문에 P_1을 $C_{1,1}$까지 롤백해야만 한다. 그러면 메시지 m4가 완료되지 않았기 때문에 P_2를 $C_{2,0}$까지 롤백해야만 한다. 이런 식으로 모든 프로세스의 초기 상태까지 롤백해야만 한다.[1]

본 논문에서는 실험을 위한 메시지 전달방식 통신 라이브러리로 MPI(Message Passing Interface)[7][8]를 사용했는데, 이것은 서로 다른 시스템간에도 프로그램 코드의 변경 없이 수행하게 하기 위해 제정된 표준으로서 많은 수퍼컴퓨터 벤더들이 지원해주는 API이기도 하며, 1994년 MPI 버전 1.0이 발표된 이후 지금까지 2.0까지 나와있다.

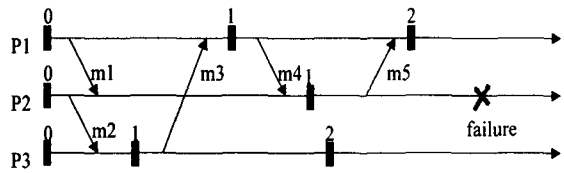

그림 1. Fault 발생 후 복구 시 발생하는 캐스캐이드 롤백. "■"는 체크포인트를 말한다. 고장을 복구하기 위하여 모든 프로세스는 자신의 초기 체크포인트까지 롤백해야만 한다.

3. 시스템 모델

본 논문에서 프로세스는 단지 메시지 전달을 통해서만 다른 프로세스와 통신을 할 수 있다고 가정한다. 그리고 시스템은 지역 기계의 고장을 발견하기 위한 부분과 프로세스의 고장을 발견하기 위한 부분이 부득이하게 타임 클락을 공유하는 동기화 한 특징을 갖지만 시스템은 전체적으로 비동기 적이다.

3.1. 서버

본 논문에서 서버는 안정적이라고 가정한다. 이것은 구현의 편의를 위한 것인데, 만약 그렇지 않다면 미러 서버를 만들어 서버의 상태와 동등하게 유지하여 주면 된다. 그러다 서버에 고장이 발생하게 되면 지금까지 유지해 왔던 미러 서버를 이용하여 서버를 원 상태로 복구하면 된다. 본 논문의 시스템 구성은 그림 2에 잘 나타나 있다.

서버는 중재자와 기계 상태 테이블로 구성되어 있다. 중재자는 각 프로세스들로부터 모든 메시지의 복사본을 받은 후, 이것을 이용하여 각 기계의 상태를 알아내어 서버에 있는 기계 상태 테이블에 저장한다. 그러면서 각 프로세스들로부터 받은 메시지를 이용하여 캐스캐이드 롤백을 유발하는 잠재적 고아 메시지를 적응형 알고리즘에 적용시켜 찾아내고 필요한 경우에는 그 프로세스에게 체크포인트를 추가하도록 한다. 또 중재자는 모든 지역 기계의 고장을 발견하기 위하여 주기적으로 각 지역 기계 안에 있는 고장 발견자(failure detector)에게 간단한 제어 메시지를 보내고, 일정한 시간 안에 응답이 오지 않으면 지역 기계 고장으로 간주한다. 이렇게 지역 기계 고장이 발견되면 서버에 유지되고 있는 기계 상태 테이블을 이용하여 고장이 발생한 지역 기계를 복구한다. 기계 상태 테이블은 중재자가 전해주는 각 지역 기계의 상태를 저장하는 역할을 한다.

3.2. 지역 기계

지역 기계는 고장 발견자 프로세스들로 구성되어있다. 고장 발견자는 자신이 속한 지역 기계 안에 있는 모든 프로세스의 고장을 발견하기 위하여 주기적으로 지역 기계 안의 모든 프로세스에게 간단한 제어 메시지를 보내고, 일정한 시간 안에 응답이 없으면 프로세스 고장으로 간주한다. 이렇게 프로세스 고장을 발견하면 메모리에 저장되어 있는 그 프로세스의 가장 최근의 체크포인트를 가져와서 고장이 발견된 프로세스를 복구하고, 가장 최근의 체크포인트부터 재시작 한다. 또 이런 작업과는 상관없이 서버의 중재자가 주기적으로 보내오는 제

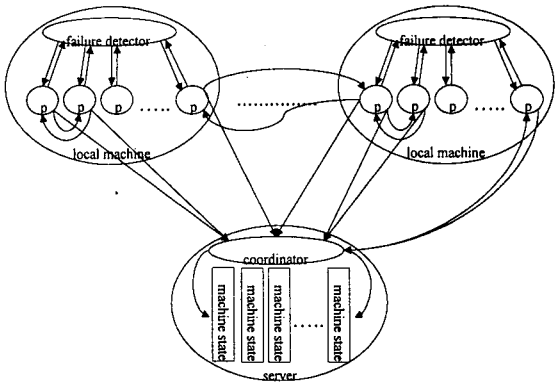

그림 2. 시스템 모델

어 메시지에 응답을 한다. 프로세스는 기본적으로 다른 프로세스들과 메시지를 주고받는 일을 한다. 그러면서 메시지를 송신할 경우에는 메시지의 복사본을 서버에 있는 중재자에게 보내게 되는데 이렇게 하면 결국에는 프로세스사이에서 오고가는 모든 메시지의 복사본이 중재자에게 보내지고 이것을 이용하여 모든 지역 기계의 상태가 서버의 기계 상태 테이블에 유지될 수 있다. 또 프로세스는 이런 작업과는 상관없이 고장 발견자가 주기적으로 보내오는 제어 메시지에 응답을 한다.

4. 구현과 성능 평가

4.1. 구현

Pentium PC를 사용하고 운영체제로는 리눅스를 사용한다. 설치한 리눅스 배포판은 래드햇(Red Hat) 6.0으로서, 커널 버전은 2.2.9이다. 각 PC들은 LAN(Local Area Network)으로 연결되어 있으며 메시지 전달방식 통신 라이브러리로 MPI(Message Passing Interface)를 사용하였다.

4.2. 성능 평가

본 논문의 방법과 기존의 적응형 체크포인팅 방법을 간단한 벤치마킹 프로그램인 핑퐁(pingpong) 프로그램을 이용하여 실험을 한다. 성능 비교 방법은 총 메시지 개수, 수행시간, 체크포인트 개수를 비교하여 나타낸다.

본 논문의 총 메시지 수는 프로세스 사이에 오고가는 메시지 수에다 각 지역 기계의 상태를 서버가 유지하기 위해서 프로세스 사이에 오고가는 모든 메시지의 복사본, 프로세스의 고장을 발견하기 위하여 고장 발견자가 자신이 속한 기계의 프로세스들과 주고받는 제어 메시지, 지역 기계의 고장을 발견하기 위하여 중재자가 각 지역 기계의 고장 발견자와 주고받는 제어 메시지, 중재자가 잠재적 고아 메시지를 발견하여 프로세스에게 체크포인트를 추가하도록 하는 메시지를 더하면 된다. 이렇게 하면 총 메시지 개수는 모든 메시지의 복사본을 서버에 보내는 방법으로 인하여 기존의 체크포인팅 방법과 비교해서 2배 정도 많아 진다.

수행시간은 본 논문이 적응형 방법을 사용하여 모든 캐스캐이드 롤백을 피하기 때문에 기존의 적응형 방법보다는 짧아진다.

체크포인트의 개수는 기존의 적응형 방법과 비교하여 비슷하게 나타난다.

5. 결론과 향후 연구 방향

비동기 체크포인팅 프로토콜은 체크포인트를 두기 위하여 전체 시스템이 정지할 필요가 없는 방법이고, 적응형 체크포인팅 알고리즘은 비동기 체크포인팅 사용 시에 어느 한 프로세스의 고장 발생이 다른 프로세스의 롤백까지 유도하는 캐스캐이드 롤백의 경우를 막아줄 수 있는 방법이다.

본 논문에서는 기본적으로 비동기 체크포인팅 프로토콜을 사용하면서 캐스캐이드 롤백을 막기 위한 적응형 체크포인팅 알고리즘을 함께 사용하였는데 기존의 방법과는 다르게 모든 경우의 예외 사항을 중재자가 발견하여 체크포인트를 추가하도록 하였다. 이렇게 하여 전체 수행시간이 짧아지게 되었고, 또한 프로세스 고장뿐만 아니라 지역 기계의 고장도 복구할 수 있도록 서버에 각 기계들의 상태를 기계 상태 테이블에 유지하였다.

앞으로는 서버에 고장이 발생했을 경우도 고려한 연구가 진행되어야 한다.

참고 문헌

[1] Jian Xu and Robert H. B. Netzer, "Adaptive Independent Checkpointing for Reducing Rollback Propagation", Dept, of Computer Science Brown University, 5th IEEE Symposium on Parallel and Distributed Processing, December, 1993.

[2] Baldoni R and Helary JM and Mostefaoui A and Raynal M, "Adaptive Checkpointing in Message Passing Distributed Systems", International Journal of Systems Science, 1997.

[3] D. B. Johnson and W. Zwaenepoel, "Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing", Journal of Algorithms 11 pp. 462-491, 1990

[4] D. B. Johnson and W. Zwaenepoel, "Sender-based Message Logging", In Proc. Conf. on Fault-Tolerant Computing Systems, pp. 14-19, 1987.

[5] Kwang-Sik Chung and Ki-Bom Kim and Chong-Sun Hwang and Jin-Gon Shon and Heon-Chang Yu, "Hybrid Checkpointing Protocol based on Selective Sender-based Message Logging", Proceedings of the 1997 International Conference on Parallel and Distributed Systems, December, 1997.

[6] R. E. Strom and S. Yemini, "Optimistic Recovery in Distributed Systems", ACM Trans. on Computer Systems 3 pp. 204-226, August, 1985.

[7] Message Passing Interface Forum. MPI : A Message Passing Interface Standard, Mar 1994.

[8] Wei-Jih Li and Jyh-Jong Tsay. "Checkpointing Message-Passing Interface(MPI) Parallel Programs", Dept. of Computer Science and Information Engineering, National Chung Cheng University, Proceedings of the 1997 Pacific Rim International Symposium on Fault Tolerant Systems, 1997.