분산 계산 환경의 검사점 작성 및 롤백 복구 프로토콜

안 성 준*, 조 유 근
서울대학교 컴퓨터공학과

Checkpointing and Rollback-Recovery Protocols in Distributed Computing Systems

Seong-Jun Ahn, YooKun Cho
Department of Computer Engineering, Seoul National University

요 약

메시지 전달을 이용한 분산 계산 환경의 검사점 작성 및 롤백 복구 프로토콜은 조정 검사점 작성(coordinated checkpointing), 약조정 검사점 작성(loosely coordinated checkpointing), 독립적 검사점 작성(independent checkpointing)등 크게 세 종류로 구분할 수 있다. 이 프로토콜들의 성능은 프로세스간 통신의 빈도, 통신의 패턴 등 응용의 특성 및 수행 환경에 영향을 받는다. 기존에 제안된 프로토콜 각각의 성능에 대해서는 많은 연구가 있었으나 이질적인 종류의 프로토콜들을 동일한 환경에서 구현하여 성능을 비교하는 연구는 이루어지지 않았다.

본 논문에서는 검사점 작성 및 롤백 복구 프로토콜들을 구현하고, 동일한 환경에서 성능을 측정한 결과를 제시한다. 아울러 검사점 작성 및 롤백 복구 프로토콜의 성능에 영향을 미치는 요소들을 분석하여, 이들 프로토콜의 성능 평가 기준과 응용의 특성에 적합한 프로토콜의 선택 기준을 제시한다.

1 서론

검사적 작성과 롤백 복구는 시스템의 프로세스들이 결함이 없는 상태로 수행(failure-free execution) 중일 때 각 프로세스의 상태를 주기적으로 안전한 저장소(stable storage)에 저장하여, 시스템에 결함이 발생하였을 때 저장된 상태에서 시스템이 다시 시작할 수 있게 한다. 즉, 시스템이 결함이 발생하더라도 시스템이 잃어버리는 작업을 최소화함으로써 시스템에 결함 허용을 제공하는 기법이다. 저장된 프로세스의 상태를 검사점(checkpoint)이라고 하고, 이전에 작성된 검사점에서 프로세스를 다시 시작하는 것을 롤백 복구라고 한다. 검사점 작성과 복구는 수행 시간이 긴 과학 응용에서는 결함이 발생하였을 때 전체 수행 시간을 최소화하는데 사용되며, 서비스의 응답성이 중요한 실시간 응용에서는 서비스의 다운 시간을 최소화하여 가용성을 높이는데 사용된다[1].

분산 계산 환경의 검사점 작성과 롤백 복구 프로토콜은 전체 시스템의 일관성 유지를 보장해야 한다. 메시지 전달을 이용한 분산 계산 환경의 검사점 작성 및 롤백 복구 프로토콜은 조정 검사점 작성, 약조정 검사점 작성, 독립적 검사점 작성 등 크게 세 종류로 구분할 수 있다. 이 프로토콜들의 성능은 프로세스간 통신의 빈도, 통신의 패턴 등 응용의 특성 및 수행 환경에 영향을 받는다. 기존에 제안된 프로토콜 각각의 성능에 대해서는 많은 연구가 있었으나 이질적인 종류의 프로토콜들을 동일한 환경에서 구현하여 성능을 비교하는 연구는 이루어지지 않았다. 본 논문에서는 이들 기법을 구현하고 여러 환경 하에서 다양한 특성을 가지는 응용으로 실험을 하여, 각 프로토콜들의 특성과 성능을 비교 분석하고 문제점을 제시한다. 아울러 검사점 작성 및 복구 프로토콜의 성능에 영향을 미치는 일반적인 요소들에 대해 분석하여, 이들 프로토콜의 성능 평가 기준과 응용의 특성에 따른 프로토콜의 선택 기준을 제시한다.

본 논문의 구성은 다음과 같다. 먼저 2 장에서 검사점 작성 방법 및 복구 프로토콜에 대해 설명하며, 3 장에서 검사점 작성 및 복구 프로토콜을 구현하여 실험 결과를 제시하고 결과를 분석한 내용을 설명한다. 마지막으로 4 장에서 결론을 맺는다.

2 검사점 작성 및 복구 프로토콜

2.1 독립적 검사점 작성

독립적 검사점 작성 프로토콜에서는 각 프로세스가 원하는 시점에서 자율적으로 검사점을 작성할 수 있다. 이 기법은 정상 수행 시간 오버헤드가 낮은 반면 도미노 효과(domino effect)[3]의 발생 가능성이 높다. 또한 각 프로세스는 다중 검사점을 유지해야 하며, 검사점이 더 이상 유용하지 않음을 선언하기 위해서 주기적으로 가비지 컬렉션(garbage collection) 알고리즘을 수행해야 한다.

복구 시에 일관된 전체 검사점을 결정하기 위해서는 정상 수행 시간 동안 메시지 교환에 의해 발생하는 검사점 사이의 종속 관계를 기록해야 한다. 실패가 발생하면 복구 개시자는 각 프로세스에서 독자적으로 유지하고 있는 종속성 정보를 종합하기 위해 종속성 요청(dependency request) 메시지를 브로드캐스팅 한다. 종속성 요청 메시지를 받은 프로세스는 수행을 중단하고 저장된 종속성 정보와 현재 상태의 종속성 정보를 복구 개시자에게 전달한다. 현재 상태의 종속성 정보를 가상 검사점(virtual checkpoint)[4]이라고 한다. 복구 개시자는 전체의 종속성 정보를 사용하여 복구선을 계산한 후 결과를 복구 요청(rollback request) 메시지에 담아 브로드캐스팅 한다. 복구 요청 메시

지를 받은 프로세스는 자신의 가상 검사점이 복구선에 포함되어 있으면 수행을 재개하고, 저장된 검사점이 복구선에 포함되어 있으면 복구선에 지정된 검사점으로 롤백 복구한다.

독립적 검사점 작성에서 가비지 컬렉션은 복구선을 계산하고, 복구선 이전 상태에 해당하는 검사점을 지우는 과정으로 이루어진다. 가비지 컬렉션을 위해 복구선을 계산할 때는 가상 검사점의 종속성 정보, 즉 프로세스의 현재 상태를 제외한 복구-종속 그래프 또는 검사점 그래프를 사용한다.

2.2 조정 검사점 작성

조정 검사점 작성(coordinated checkpointing)에서는 검사점 작성시 시스템의 일관된 상태를 위하여 모든 프로세스가 조정에 참여한다. 조정에 의해 작성된 검사점은 일관된 상태를 형성한다. 이 기법에서는 모든 프로세스가 가장 최근의 검사점에서 수행을 재개하므로 도미노 효과가 발생하지 않는다. 또한 복구와 가비지 컬렉션 과정이 매우 간단하며, 프로세스마다 하나의 검사점을 유지하면 되므로 저장장치의 오버헤드가 독립적 검사점 작성보다 작다. 반면에 각 프로세스가 필요한 때에 독자적으로 검사점을 작성할 수 없다는 단점이 있으며, 응용 프로그램에서 필요한 메시지 외에 검사점 작성 조정을 위한 메시지 오버헤드가 추가된다는 단점이 있다.

Koo 와 Toueg[2]은 다음과 같은 이단계 프로토콜을 제안하였다. 첫 번째 단계에서 검사점 조정 개시자는 마지막 검사점 작성 이후 통신했던 프로세스들에게 검사점 요청 메시지를 전달한다. 요청 메시지를 받은 프로세스는 마지막 검사점 이후 자신과 통신했던 모든 프로세스에게 검사점 요청 메시지를 보낸다. 이 과정은 종속 관계에 있는 모든 프로세스가 알려질 때까지 재귀적으로 반복된다. 두 번째 단계에서 개시자와 종속 관계에 있는 모든 프로세스는 검사점을 작성한다. 프로토콜의 결과는 프로토콜에 참여한 프로세스만을 포함하는 일관된 검사점이다. 프로토콜 진행 동안에는 프로세스간 통신이 봉쇄되어야 한다. 시스템에 실패가 발생하면 모든 프로세스는 가장 최근에 커밋된 검사점으로 롤백 복구한다.

2.3 약조정 검사점 작성

약조정 검사점 작성, 또는 통신 유도 검사점 작성(communication-induced checkpointing)은 독립적 검사점 작성의 도미노 효과를 피하는 기법으로, 복구선의 전진을 보장하기 위해 검사점 작성과 통신 패턴에 대한 시스템 전반적인 제약 조건이 명세 된다. 충분한 정보가 메시지마다 덧붙여지며, 수신자는 메시지를 처리하기 전에 덧붙여진 정보를 먼저 검사할 수 있다. 만약 메시지를 처리하는 것이 명세 되어진 제약 조건을 위반하는 경우, 수신 프로세스는 메시지를 처리하기 전에 검사점을 작성해야 한다.

이 프로토콜에서는 조정 검사점 작성과는 달리 프로세스간 조정을 위한 다른 별다른 메시지가 필요 없다. 그러나 복구를 위해 유지해야 하는 검사점의 수가 많기 때문에 저장 오버헤드가 크고, 주기적으로 가비지 컬렉션을 해줘야 하는 단점이 있다. 실패가 발생했을 때의 복구 과정과 가비지 컬렉션 과정은 독립적 검사점 작성과 동일하다.

3 실험 및 결과

3.1 실험 환경

분산 계산 환경을 구축하기 위해 4 대의 Pentium 200 을 장착한 PC 를 Ethernet 을 사용하여 연결하였다. 운영체제는 Linux(커널 2.0.35)를 사용하였으며, 이 커널에 검사점 작성 및 복구 기능을 추가하였다. 메시지의 전달은 Unix socket 의 UDP 를 사용하였으며, 안정적인 메시지의 전달을 위해 흐름제어 기능과 선입선출(FIFO) 기능을 응용 프로그램 수준에서 구현하였다. 검사점은 각 호스트의 하드 디스크에 저장되도록 하였다.

3.2 응용 프로그램

검사점 작성 기법의 성능은 실행되는 응용 프로그램의 특성에 많은 영향을 받는다. 성능에 영향을 미치는 응용의 특성으로는 메모리 사용량, 통신 유형, 통신 빈도, 메시지 크기 등이 있다. 메모리 사용량은 저장되는 검사점의 크기와 밀접한 관련이 있다. 사용하는 메모리가 클수록, 검사점 작성 오버헤드가 커진다. 수행 시간이 오래 걸리는 대부분의 분산 응용들은 문제의 크기가 크기 때문에 많은 메모리를 차지하는 경향을 보인다. 통신 유형은 프로세스간 종속성에 영향을 미친다. 통신의 대상이 여러 프로세스로 분산될 수록 종속 관계가 복잡해져서 일관성을 유지하기 위해 작성하는 검사점의 수를 증가 시킨다. 통신 빈도는 단위 시간당 보내는 메시지의 수를 의미한다. 실험에 사용한 응용의 특성은 다음과 같다.

- CF : 2048 x 2048 행렬에 대해 Cholesky Factorization 을 수행한다. 여러 개의 프로세스가 번갈아 가며 메시지를 브로드캐스팅 하는 통신 유형을 보이며, 초당 약 20 개의 메시지를 주고받는다. 사용하는 메모리는 약 3.5 Mbytes 이다. 통신 빈도가 제일 높은 응용이다.
- MM : 두 개의 1000 x 1000 행렬을 곱하는 연산을 수행한다. 처음에 행렬을 각 프로세스에게 나누고, 결과를 종합하는 외에는 통신을 하지 않는다. 사용하는 메모리는 약 11 Mbytes 이다. 통신 빈도는 제일 낮으며 사용하는 메모리는 가장 큰 응용이다.
- RD : 2 초 간격으로 256 바이트의 메시지를 임의의 프로세스에게 전송한다. 사용하는 메모리는 1.1 Mbytes 이다. 메모리 사용량은 가장 적으며 통신 빈도는 중간이다.

3.3 실험 결과 및 분석

이 절에서는 위에서 설명한 응용에 각 검사점 기법을 적용하여 실행한 결과를 제시하고 분석한다. 각 응용은 네 개의 프로세스에 의해 분산적으로 수행되었고, 각각의 호스트마다 하나의 프로세스가 수행되었다. 모든 실험에서 CF 는 검사점 작성 주기를 20 초로 했으며, MM 은 30 초, RD 는 20 초로 했다. 약조정 검사점 작성과 독립적 검사점 작성에서 가비지 컬렉션를 실행하는 주기는 검사점 작성 주기의 두 배로 했다. 표 1 과 표 2 는 실험 결과를 보여준다. 표 1 의 결과는 세 가지 응용을 각각의 프로토콜을 적용하여 실행했을 때의 수행시간을 초 단위로 나타낸 것이며, 표 2 의 결과는 검사점 작성을 하지 않았을 때에 비해 증가된 수행시간의 비율을 퍼센트 단위로 나타낸 것이다.

3.3.1 검사점 작성 오버헤드

검사점 작성으로 인해 증가하는 수행시간을 응용별로 고찰하면 다음과 같다. 응용 CF 의 수행 결과를 보면, 수행시간은 236 초에서 272 초의 분포를 보인다. 각 프로토콜 별로 살펴보면 독립적 검사점 작성이 가장 좋은 성능을 나타냈고, 조정 검사점 작성의 성능이 가장 좋지 않았다. 조정 검사점 작성의 경우 작성된 검사점의 개수는 프로세스당 평균 17 개이다. 그러나 약조정 검사점 작성과 독립적 검사점 작성은 평균 14 개의 검사점을 작성하였다. 조정 검사점 작성에서 검사점의 수가 많은 것은 동시에 여러 프로세스가 검사점 작성을 개시하려고 하기 때문이다. 검사점을 커밋 할 수 있음에도 불구하고 다른 조정자와의 간섭 때문에 취소된 검사점의 수는 프로세스당 3 개이다. 따라서 이러한 경우가 발생하지 않는다면 다른 기법과 동일한 수의 검사점을 작성하게 될 것이다. 조정 검사점 작성의 오버헤드가 높은 것은 많은 수의 검사점을 작성한 것이 주 원인이며, 수행 시간에 검

사점 작성 조정 시간과 검사점 작성 조정 중 메시지를 보내지 못하고 봉쇄된 시간도 오버헤드의 원인이 된다. 검사점 작성 조정 시간은 프로세스 당 평균 3.2 초이며, 메시지 봉쇄 시간은 약 0.4 초로 오버헤드의 10 퍼센트를 차지한다. 약조정 검사점 작성과 독립적 검사점 작성의 오버헤드는 검사점 작성 시간과 가비지 컬렉션 시간으로 이루어진다. 약조정 검사점 작성의 경우 가비지 컬렉션에 걸린 시간은 약 1.2 초이며 독립적 검사점 작성의 경우에는 1.3 초이다. 나머지 오버헤드는 검사점 작성의 오버헤드이다.

MM 의 수행 결과를 CF 의 경우와 비교해 보면 전체적으로 검사점 작성의 오버헤드가 큰 것을 알 수 있다. 이것은 프로세스가 사용하는 메모리가 크기 때문에, 검사점 생성 시간이 길기 때문이다. 반면 프로토콜 간의 차이는 그다지 크지 않다. 이유는 MM 이 메시지를 거의 주고 받지 않기 때문이다. MM 은 시작될 때 행렬을 나누고, 결과를 조합하는 이외에는 메시지를 주고 받지 않는다. 약조정 검사점 작성과 독립적 검사점 작성의 수행 시간이 조정 검사점 작성보다 오래 걸리는 까닭은 주기적인 가비지 컬렉션 작업 때문이다.

응용 RD 의 검사점 작성 오버헤드는 MM 의 경우와 마찬가지로 프로토콜 간 큰 차이를 보이지 않는다. 오버헤드의 차이가 없는 이유는 응용에서 사용하는 메모리가 작기 때문에 검사점 생성 시간이 작기 때문이다.

응용 프로그램	검사점 없음	독립적 검사점	조정 검사점	약조정 검사점
CF	236	264	272	265
MM	348	538	515	538
RD	300	305	304	306

<표 1. 수행시간 측정 결과 (초)>

응용 프로그램	독립적 검사점	조정 검사점	약조정 검사점
CF	11.86	15.25	12.29
MM	54.6	47.99	54.6
RD	1.67	1.33	2.0

<표 2. 오버헤드 측정 결과 (%)>

3.3.2 복구 오버헤드

시스템에 실패가 발생한 후 수행을 재개하기까지 걸린 시간을 살펴보면 다음과 같다. CF 의 복구 시간은 조정 검사점 작성이 0.83 초, 약조정 검사점 작성이 1.15 초 독립적 검사점 작성이 1.21 초로 프로토콜 간에 큰 차이를 보이지 않는다. MM 의 복구 시간은 조정 검사점 작성이 3.1 초 약조정 검사점 작성과 독립적 검사점 작성이 3.3 초로 거의 동일한 결과를 보였다. RD 의 복구 시간은 모두 0.7 초 내외로 동일한 결과를 보였다. 따라서 복구시간은 프로토콜 간에 큰 차이가 없음을 알 수 있다.

다음은 복구 오버헤드를 고찰한다. 복구 오버헤드는 실패로 인하여 잃어버리는 계산의 양, 즉 롤백 거리를 의미한다. 응용 CF 에서 조정 검사점 작성을 수행할 경우 롤백 거리는 평균 9.7 초로 검사점 작성 주기의 반 정도이다. 약조정 검사점 작성의 경우는 13.5 초이며 독립적 검사점 작성은 14.3 초이다. 약조정 검사점 작성과 독립적 검사점 작성의 복구 오버헤드가 큰 것은 검사점 작성 오버헤드가 작은 것에 대한 교환 조건이라고 생각할 수 있다. 독립적 검사점 작성의 경우 복구 오버헤드에 대한 제한이 없기 때문에 도미노 효과가 발생할 수도 있으나, 실험에서 실제로 도미노 효과는 발생하지 않았다. 이는 각 프로세스가 검사점을 작성하는 시점이 거의 동일하기 때문이다. 응용 MM 의 복구 오버헤드도 세 기법이 거의 동일하게 15 초 내외의 결과를 보였다. 이는 응용의 통신 빈도가 낮기 때문이다. 응용 RD 의 복구 오버헤드 역시 모든 기법이 약 8.2 초로 유사한 결과를 보였다. RD 에서 복구 오버헤드가 작은 것은 프로세스간 검사점 작성 주기와 메시지를 전달하는 시점이 동기화 되기 때문이다.

3.3.3 분석

실험 결과를 종합해 보면 다음과 같은 사실을 알 수 있다. 검사점 작성 오버헤드에 가장 큰 영향을 미치는 것은 검사점의 개수이다. 생성되는 검사점의 수는 통신 빈도에 영향을 받게 되는데, 통신 빈도와 검사점 작성 주기의 비가 작을 수록 작성되는 검사점의 수가 줄어든다. 복구 시간은 검사점 작성 오버헤드와 비교해 볼 때 극히 작으며, 프로토콜의 성능에 큰 영향을 미치지 않는다. 복구 오버헤드는 조정 검사점 작성이 가장 작으며, 독립적 검사점 작성의 복구 오버헤드가 가장 크다. 그러나 프로세스간 검사점 작성 주기가 거의 일치하고 메시지를 보내는 시점이 검사점 작성 시간과 동기화 되면 복구 오버헤드가 상당히 줄어든다.

통신 빈도가 높은 응용은 독립적 검사점 작성과 약조정 검사점 작성 프로토콜을 사용하는 것이 좋은 성능을 나타낸다. 프로세스마다 검사점을 작성하는 시점이 일치할 때에는 독립적 검사점 작성이 약조정 검사점 작성보다 성능이 좋다. 통신 빈도가 낮은 응용은 조정 검사점 작성 프로토콜을 사용하는 것이 오버헤드가 가장 적다.

4 결론

본 논문에서는 메시지 전달 시스템에서 결함을 허용하기 위해 사용하는 주요 기법인 검사점 작성 및 복구 기법에 대해 조사하여 각 기법의 특성과 장단점을 알아보았다.

조정 검사점 작성은 매 검사점 마다 프로세스간 조정 작업을 해야 하기 때문에 검사점 작성 오버헤드가 크지만, 복구 오버헤드가 작은 특성을 가진다. 약조정 검사점 작성은 검사점 작성 오버헤드는 작지만 복구 오버헤드, 즉 롤백 거리가 길어지는 특성을 가지고 있으며, 독립적 검사점 작성은 순수한 검사점 생성 이외의 검사점 작성 오버헤드는 전혀 없으며, 도미노 효과가 발생할 수 있다.

또한 본 논문에서는 실제로 이들 기법을 구현하고 실험하여 성능을 평가한 결과를 제시하였다. 응용의 통신 빈도가 높을 때에는 조정 검사점 작성의 검사점 작성 오버헤드가 가장 크고 독립적 검사점 작성의 검사점 작성 오버헤드가 가장 작으며, 복구 시간은 거의 동일하다. 복구 오버헤드는 조정 검사점 작성이 최소이며 독립적 검사점 작성이 최대이다. 프로세스간 검사점 작성 주기와 메시지 전달 시점의 동기화 여부가 복구 오버헤드에 영향을 미치는 주요 요소이다.

[참고 문헌]

[1] E. N. Elnozahy, D. B. Johnson, and Y. M. Wang, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems", Technical Report CMU-CS-96-101, Carnegie Mellon University, 1996.

[2] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems", IEEE Trans. Software Eng., SE-13(1):23-31, January 1987.

[3] B. Randell, "System structure for software fault tolerance", IEEE Trans. Software Eng., SE-1(2):220-232, June 1975.

[4] Y. M. Wang and W. K. Fuchs, "Lazy checkpoint coordination for bounding rollback propagation", In Proc. IEEE Symp. Reliable Distributed Syst., pp. 78-85, October 1993.