다중 분할된 구조를 가지는 클러스터 검사점 저장 기법

장 윤 석†

요 약

검사점 저장 기법을 사용하여 주기적으로 클러스터 노드들의 프로세스 수행 정보를 전역 저장 장치에 저장하는 분산 클러스터 시스템에서 결함 허용 성능을 유지하는 데 드는 비용을 줄이고 전체 프로세스의 수행 성능을 증가시키기 위해서는 검사점 정보를 저장할 때에 네트워크로 전달되는 부하를 각 노드에 최대한 적절하게 분산하여 데이터 저장 시간을 줄임으로써 검사점 정보를 저장하는 동안 전체 클러스터 시스템의 프로세스가 지연되는 시간을 줄이도록 하여야 한다. 이를 위하여 분산 RAID 기반의 단일 입출력 공간을 사용하는 클러스터 시스템에서는 여러 가지 검사점 저장 기법을 사용하며, 검사점 정보의 저장 기법에 따라서 저장 성능과 결함 회복 성능이 달라진다. 본 연구에서는 분할된 검사점 저장 기법을 개선하여 검사점 데이터를 분산 RAID 기반의 단일 입출력 공간에 저장할 때에 그룹별로 분할되는 분할 그룹 크기를 검사점 정보가 저장될 때의 네트워크의 트래픽에 따라서 동적으로 결정하여 네트워크를 통한 분산 RAID에 저장함으로써 네트워크 병목현상을 최소화하는 다중 분할된 검사점 저장 구조를 제안하였다. 제안된 구조의 성능을 분석하기 위하여 최대 512개의 가상 노드로 구성된 클러스터 시스템을 대상으로 하여 MPI 와 Linpack HPC 벤치마크를 통한 성능 평가를 수행하였으며, 성능 평가 결과는 검사점 정보의 크기와 클러스터의 크기가 증가할수록 제안된 기법이 검사점 정보의 저장과 결함 회복 능력에 대하여 기존의 검사점 저장 기법에 비하여 우수한 성능을 보인다.

키워드 : 다중 분할된 검사점 저장 기법, 검사점 정보, 클러스터 컴퓨터, 분산 RAID, 결함 허용

A Multistriped Checkpointing Scheme for the Fault-tolerant Cluster Computers

Yun Seok Chang†

ABSTRACT

The checkpointing schemes should reduce the process delay through managing the checkpoints of each node to fit the network load to enhance the performance of the process running on the cluster system that write the checkpoints into its global stable storage. For this reason, a cluster system with single IO space on a distributed RAID chooses a suitable checkpointng scheme to get the maximum IO performance and the best rollback recovery efficiency. In this paper, we improved the striped checkpointing scheme with dynamic stripe group size by adapting to the network bandwidth variation at the point of checkpointing. To analyze the performance of the multi striped checkpointing scheme, we applied Linpack HPC benchmark with MPI on our own cluster system with maximum 512 virtual nodes. The benchmark results showed that the multistriped checkpointing scheme has better performance than the striped checkpointing scheme on the checkpoint writeing efficiency and rollback recovery at heavy system load.

Key Words : Multistriped Checkpointing Scheme, Checkpoint, Cluster Computer, Distributed Raid, Fault-tolerant

1. 서 론

최근의 클러스터 시스템은 고가용성(High availability)과 고신뢰성(High reliability), 그리고 고성능(High performance)라는 사용자 요구 조건을 최대한 충족시킬 수 있는 구조를 바탕으로 구현되고 활용되고 있다. 결함 허용(Fault-tolerant) 구조는 이와 같은 요구 조건을 충족시키는 데에 있어서 가장 기본적으로 지원되어야 할 시스템 구조라고 할 수 있다. 일반적으로 클러스터 시스템을 구성하는 다수의 노드들이 매우 큰 부하를 가지는 응용 프로세스를 용이하게 완료시킬 수 있도록 하기 위하여 각 노드에서 동시에 실행되는 모든 프로세스들은 메시지 전송을 통하여 상호간에 통신을 수행함으로써 동기적인 프로세스 수행이 가능하도록 한다[9]. 이 경우, 하나 이상의 노드에서 결함이 발생될 경우에 클러스터 차원에서 최소한의 비용으로 결함 회복(Rollback recovery)을 수행하기 위하여, 각 노드들은 자신이 수행하고 있는 프로세스에 대한 정보, 즉 결함 허용 정보를 안정된 전역 저장 장치에 주기적으로 저장하거나 노드간 공유 데이터에 대한 동기화를 수행함으로써 전역적인 일

※ 본 연구는 2005년도 대진대학교 학술연구과제로 수행되었습니다.
† 종신회원 : 대진대학교 공과대학 컴퓨터공학과 교수
논문접수 : 2006년 6월 28일, 심사완료 : 2006년 10월 16일

관성을 유지할 수 있도록 하여야 한다. 이를 위하여 로그 기반의 동기화 기법이나 결함 회복 알고리즘과 같은 여러 가지 회복 기법들이 사용되고 있다.

로그 기반 복귀 회복 기법은 분산 시스템에서 결함 포용성을 제공하는 기법이다[17]. 가족 기반 메시지 로깅 프로토콜은 결함이 회복될 때에 남아 있는 프로세스들이 자신의 프로세스를 계속 수행하지 못하도록 한다[1]. Manetho 회복 알고리즘은 나머지 프로세스들이 회복 프로세스로부터 메시지를 수신한 후 결함 회복 정보를 전역 저장 장치에 저장하도록 하고, 각 프로세스들의 상태가 결함 회복 프로세스의 상태와 일관적이지 못하게 만들 수 있는 모든 프로세스들로부터의 메시지 전달을 결함에서 회복되는 프로세스가 완전히 회복될 때까지 지연시킨다[7]. 동시에 Elnozahny는 전역 저장 장치에 대한 접근 횟수를 줄이고 프로세스들에서 동시적으로 결함이 발생하는 경우에도 계속적으로 자신의 프로세스를 수행할 수 있는 회복 알고리즘을 제안하였다[6]. 그러나 이 알고리즘이 비동기적인 검사점 저장 기법(Checkpointing scheme)과 함께 구현될 경우에는 복수의 프로세스들에 결함이 발생될 경우 시스템의 일관성이 유지되지 못할 가능성이 있다는 약점이 있다.

이와 같은 결함 회복 기법들은 기본적으로는 로그 메시지나 결함 허용 정보들을 저장 장치에 미리 저장하는 특성을 가지고 있다. 특히 검사점 저장 기법이 최근의 결함 허용 기법에서 많이 적용되는 것을 고려할 때에, 클러스터 시스템의 결함 회복 성능은 클러스터 저장 장치의 구조와 클러스터 저장 장치로 결함 허용 정보를 저장하는 저장 기법에 많은 영향을 받게 된다. 초기의 클러스터 시스템과는 달리 최근의 클러스터 시스템 저장 장치들은 디스크 자체보다는 저장 서버의 입출력 대역폭 부족으로 병목현상이 일어나는 경우가 많기 때문에 다중 서버를 이용하여 입출력 병목을 해결함으로써 분산 저장 장치의 성능을 높이는 연구들이 추진되었다[2, 5, 15]. 특히 클러스터 파일 시스템은 클러스터 시스템을 구성하는 노드들에 연결된 지역 디스크들을 이용하여 분산 RAID 기반의 단일 입출력 공간(SIOS : Single IO Space)을 구축함으로써 고성능 병렬 디스크들에 근접하는 성능을 내면서도 저렴한 비용으로 구축할 수 있으므로 근래에 들어서는 과학계산용 뿐만 아니라 상업용으로도 고성능과 저가격의 이점을 바탕으로 많이 구현되고 있고, NFS에 비하여 성능이 높은 것으로 알려지고 있다[12, 16]. 이와 같은 분산 RAID 기반의 클러스터 저장 장치를 구성하는 파일시스템으로는 패리티 디스크 방식을 통하여 시스템의 가용성을 높이는 Zebra 파일시스템[8]과 공유 디스크 환경의 여러 특성을 고루 수용하여 구현된 IBM의 GPFS[17]가 그 대표적인 예이다.

이와 같은 클러스터 저장 장치를 바탕으로 클러스터 컴퓨터 시스템의 결함 허용 구조를 제공하기 위하여 여러 가지 검사점 정보 저장 기법들이 연구된 바 있다. 검사점 정보 저장 기법은 기본적으로 각 노드의 결함 허용 정보, 즉 검사점 정보(Checkpoint)를 일정한 검사점마다 안정된 클러스터 저장 장치에 저장하여 두고, 노드에 결함이 발생되면 이전의 검사점에서 저장된 결함 허용 정보를 로드하여 계속적으로 프로세스가 수행될 수 있도록 하는 방법으로 동시적 검사점 저장 기법(Coordinated checkpointing scheme)이나 순차적 검사점 저장 기법(Staggered checkpointing scheme), 분할된 검사점 저장 기법(Striped checkpointing scheme)등이 연구되고 있다. 이들 검사점 저장 기법들은 클러스터를 구성하는 모든 노드들에 대하여 전역적인 일관성을 유지하기 용이하다는 장점을 가지고 있으나, 주기적으로 검사점에서 모든 노드들에 대하여 검사점 정보를 저장하여야 하기 때문에 클러스터 저장 장치에 대한 입출력 부하가 증가되고, 특히 분산 RAID 기반의 네트워크 저장 장치를 사용하는 경우에는 네트워크 부하에 따라서 검사점에서의 입출력 처리 시간이 증가되어 전체적인 프로세스의 처리시간이 길어지는 문제점을 가지고 있다.

본 연구에서는 분산 RAID를 기반으로 하는 클러스터 저장 장치에 검사점 정보를 저장하는 방식의 검사점 저장 기법에서 각 노드들의 검사점 정보를 저장할 때에 네트워크의 부하에 따라서 클러스터 저장 장치에 대한 입출력 부하를 조정함으로써 입출력 대역폭에 의한 병목현상을 줄이고, 검사점에서의 정보 저장 시간을 줄임으로써 전체 프로세스 수행 시간을 가능한 한 줄이고자 하였다. 이를 위하여 본 연구에서는 기존의 검사점 기법 중에서 가장 우수한 입출력 성능을 가지고 있는 분할된 검사점 저장 기법을 개량하여 한 검사점에서 동시에 검사점 정보를 저장하는 노드들의 수, 즉 그룹(Stripe)의 크기를 다중 분할하여 네트워크 부하에 따라서 최대 대역폭에 알맞게 동적으로 조절함으로써 병렬 입출력 성능을 높이고, 결과적으로 검사점에서 걸리는 시간을 최소화하도록 하였다. 즉 이전의 연구에서는 그룹의 크기가 미리 결정되어 병렬 입출력이 수행되는 정적(Static)인 방식이지만, 본 연구에서는 네트워크 이용 가능 대역폭에 따라서 동적(Dynamic)으로 그룹의 크기를 결정할 수 있도록 한다. 일반적으로 동적인 구성 방식은 정적인 구성 방식에 비하여 알고리즘이 복잡하고 동작 오버헤드가 증가되는 결점이 있지만 본 연구에서 제안하는 방식은 간단한 알고리즘으로 오버헤드를 최소화함으로써 효율적인 검사점 저장을 수행할 수 있도록 하고, 이를 실험을 통하여 증명하였다.

본 논문의 구성은 다음과 같다. 2장에서는 병렬 입출력을 수행하는 기존의 검사점 기법들의 특징과 입출력 성능에 대하여 기술하였다. 3장에서는 본 논문에서 제시하는 다중 분할된 검사점 기법을 기술하고 기존 검사점 저장 기법들과의 입출력 성능 관점에서 비교하였다. 4장에서는 다중 분할된 검사점 저장 기법의 성능 평가를 수행하기 위한 시뮬레이션 환경과 시뮬레이션 도구에 대하여 기술하였다. 5장에서는 기존의 검사점 저장 기법들과 다중 분할된 검사점 기법에 대하여 수행된 시뮬레이션 결과를 비교 평가하고 실제로 구현할 경우에 발생될 수 있는 문제점과 해결 방안을 분석하였으며 6장에서 결론과 차후의 연구 방향을 기술하였다.

2. 관련 연구

클러스터 저장 장치를 사용하는 클러스터 시스템에서 결함 허용 구조를 제공하기 위하여 사용되는 검사점 저장 기법으로는 현재까지 동시적 검사점 저장 기법, 순차적 검사점 저장 기법, 그리고 분할된 검사점 저장 기법 등이 연구되고 있다. 이들 검사점 저장 기법들은 본래는 NAS Network Attached Storage)와 같이 네트워크에 연결된 하나의 전역 저장 장치를 대상으로 연구된 것이지만, 클러스터를 구성하는 각 노드에 연결되어 있는 지역 디스크들을 사용한 분산 RAID 저장 장치를 전역 저장 장치로 사용하는 클러스터 시스템에서도 동일하게 적용될 수 있다. 이 경우에는 RAID 저장 공간은 단일 입출력 공간(Single IO Space)으로 구성함으로써 각 노드들에는 하나의 독립된 저장 공간으로 사용될 수 있다[10, 11].

동시적 검사점 저장 기법은 검사점에서 모든 노드들이 동시에 검사점 정보를 전역 저장 장치에 저장하는 방법으로 각 검사점마다 모든 노드들에 대한 일관성을 유지할 수 있다는 장점을 가지고 있다[3]. 그러나 매 검사점에서 노드들의 검사점 정보들이 저장되는 동안 프로세스들의 수행이 중단될 뿐만 아니라, 다수의 노드로부터 동시에 입출력 요구들이 발생되므로 네트워크 부하가 급격하게 증가되어 입출력 병목 현상이 발생되는 문제점을 가지고 있다. 이러한 문제점을 해결하기 위하여, 디스크를 사용하지 않는 검사점 저장 기법(Diskless checkpointing scheme)이 제안되기도 하였다[14]. 그러나 디스크를 사용하지 않는 검사점 저장 기법은 특정한 시스템 구조 하에서 제한된 복구 성능만을 제공할 수 있기 때문에 동시적 검사점 저장 기법이 가진 문제점을 해결하기는 어렵다.

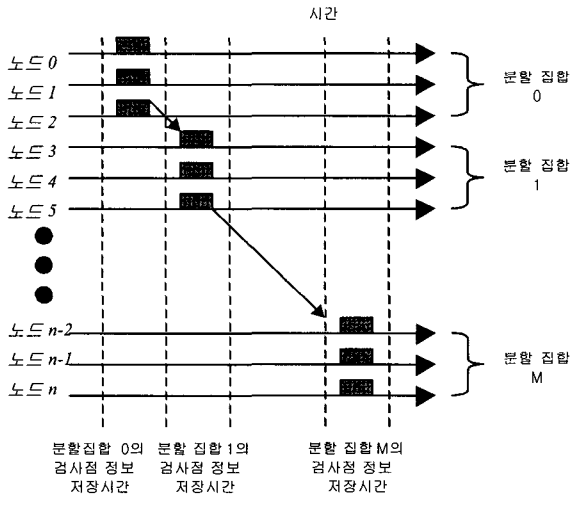
순차적 검사점 저장 기법은 검사점에서 각 노드들이 차례로 클러스터 저장 장치에 검사점 정보를 저장하므로 입출력 부하를 줄이고 네트워크 병목 현상을 방지할 수 있는 장점을 가지고 있다[18]. 그러나 이 기법은 검사점에서 정보들이 저장되는 동안 노드들 사이에 일관성이 유지되지 않는 문제점을 가지고 있기 때문에 노드들 사이에 메시지 전달을 기록하기 위한 추가적인 입출력 오버헤드를 필요로 한다는 문제점을 가지고 있다.

동시적 검사점 저장 기법과 순차적 검사점 저장 기법은 각기 입출력의 병렬성과 네트워크 병목현상 방지라는 서로 다른 특성을 제공한다. 분할된 검사점 저장 기법은 이들 검사점 저장 기법들의 특성을 부분적으로 결합하여 네트워크 병목현상을 줄이면서 병렬 입출력에 의한 이득을 최적화한다[4]. 이 기법은 전체 노드들을 여러 개의 분할 집합(Stripe group)으로 분할하고, 검사점에서 한 분할 집합 내에 포함된 노드들에 대한 검사점 정보들을 동시에 분산 RAID에 저장하면서, 각 분할 집합들의 관점에서는 순차적으로 저장한다. 즉, 각 분할 집합 내에서는 동시적 검사점 저장 기법이 적용되지만, 분할 집합들 사이에는 순차적 검사점 저장 기법이 적용된다. 이와 같은 검사점 저장 기법은 노드간의 입출력 경쟁이 매우 큰 경우에도 분할 집합 내에서 수행하는 병렬 입출력을 통하여 저장 장치 내의 입출력 대기 부하를 줄일 수 있다. 또한 노드가 매우 많을 경우에도 일정한 크기의 병렬 입출력을 수행하면서 순차적으로 입출력을 수행함으로써 네트워크의 병목현상을 방지할 수 있다. 그러므로 클러스터를 구성하는 노드의 수가 적은 경우에는 병렬 입출력의 효과 증대에 중점을 두고, 노드의 수가 많은 경우에는 네트워크 병목현상을 줄이는 데에 중점을 두어 입출력 성능을 극대화시킬 수 있다.

3. 다중 분할된 검사점 정보 저장 기법

분산 RAID의 병렬 입출력 능력은 클러스터 시스템에서 검사점 정보를 빠르게 저장하는 데에 효과적으로 이용될 수 있다. 분할된 검사점 저장 기법에서는 검사점 정보들을 분할 집합(Stripe)을 형성하고 있는 디스크들에 분산 저장할 수 있다. 하나의 분할 집합은 병렬적으로 접근할 수 있는 디스크들의 부분집합으로 정의된다. 동일한 분할 집합 내에 포함된 디스크들은 검사점 정보 저장을 위한 입출력 동작을 병렬로 동시에 실행할 수 있다. 또한 네트워크의 부하를 줄이기 위해서, 각 분할 집합들은 검사점 정보 저장을 위한 입출력 동작을 순차적으로 수행한다. 따라서 각 분할 집합들에 대한 입출력은 병렬 입출력을 수행하면서도 순차 입출력을 이용하여 입출력 병목현상을 줄일 수 있게 된다.

분할된 검사점 저장 기법에서 입출력의 병렬성을 최대로 이용할 경우에는 분할 집합의 크기가 클러스터의 크기가 되며, 이 경우는 동시적 결함허용정보저장 기법과 동일한 구조가 된다. 또한 분할 집합의 크기를 최소, 즉 1로 하면 순차적 결함허용정보저장 기법과 동일한 구조가 된다. 따라서 분할된 결함허용정보저장 기법은 병렬 입출력과 순차 입출력의 장점을 병용한 기법이라고 할 수 있다. (그림 1)은 이와 같은 분할된 검사점 저장 기법의 개념을 나타내고 있다.



(그림 1) 분할된 검사점 저장 기법의 동작 과정

이와 같은 분할된 검사점 저장 기법은 네트워크의 이용률과 입출력 성능을 모두 향상시킬 수 있다. 여기에서, 분할 집합의 크기는 프로세스가 시작되기 전에 가변적으로 결정될 수 있기 때문에, 프로세스가 수행될 때에 발생되는 부하의 크기를 미리 파악할 수 있는 특정한 응용 프로그램에 대하여, 사용자는 상대적인 분할 집합의 크기와 수를 가장 효과적인 입출력 성능을 나타낼 수 있도록 조정할 수 있다.

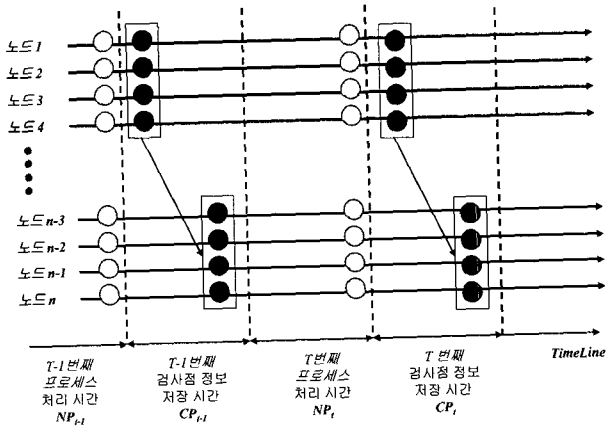
그러나 분할된 검사점 저장 기법은 프로세스가 수행되기 전에 분할 집합의 크기를 결정하기 때문에 프로세스가 수행되고 있는 도중에 전체 클러스터의 부하나 네트워크의 부하가 달라질 경우, 동적으로 적응할 수 없다는 결점을 가지고 있다. 즉 매번 검사점에서 네트워크 자체의 부하와 사용 가능한 네트워크 대역폭이 초기에 가정한 값과 달라지는 경우에는 초기에 설정한 분할 집합의 크기가 더 이상 최적의 입출력 성능을 보장하기 어렵다. 이는 클러스터를 구성하는 네트워크의 부하가 MPI에 의한 통신만을 포함하는 것이 아니라 각 노드들 간의 RPC 통신 부하, 운영체제들이 발생하는 네트워크 트래픽, 외부 네트워크로부터의 부하 등 여러 통신 부하들을 포함하고 있기 때문이다. 클러스터를 연결하는 네트워크의 부하가 매 검사점마다 달라진다면 클러스터 응용 프로세스들이 사용 가능한 네트워크 대역폭도 변화될 수밖에 없으며, 그 결과 네트워크 대역폭과 노드 수에 따라서 설정된 분할 그룹은 더 이상 최적의 성능을 나타내기 어렵게 된다. 따라서 매 검사점마다 사용 가능한 네트워크 대역폭에 대하여 가장 최적의 입출력 부하를 발생시키는 분할 그룹의 크기가 매번 새로 정의될 필요가 있다. 이를 위하여 본 논문에서는 분할 그룹 크기를 프로세스 수행 중에도 매 검사점에서 네트워크 부하에 따라서 동적으로 설정할 수 있는 다중 분할된 검사점 저장 기법(Multi striped checkpointing scheme)을 설계하였다.

다중 분할된 검사점 저장 기법에서는 각 검사점에서 분할 그룹이 검사점에서의 네트워크 부하에 따라서 여러 가지 하위 크기로 다중 분할될 수 있도록 한다. 예를 들어 임의의 프로세스의 초기 분할 그룹의 크기가 8이라면 이 프로세스의 하위 분할 그룹의 크기는 4, 2, 1 등으로 다중 분할될 수 있다. 노드수나 초기 분할 그룹의 크기가 홀수인 경우에는 마지막 하나의 노드는 별도로 처리된다. 클러스터 시스템은 각 검사점에 도달하였을 때 네트워크 부하와 사용 가능 대역폭을 측정하고, 대역폭에 적합한 하위 크기로 분할 그룹을 재구성하여 각 노드들의 검사점 정보를 저장한다. 이 과정은 다음과 같이 나타낼 수 있다.

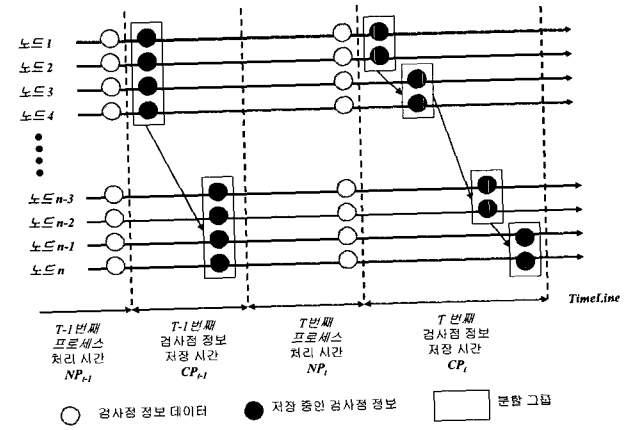
① 네트워크의 사용 가능 대역폭을 측정하고, 초기 분할 그룹 크기를 결정한 다음, 프로세스를 수행한다.
② 검사점에 도달하면 다시 사용 가능한 네트워크 대역폭을 측정한다.
③ 대역폭에 적합한 분할 그룹 크기를 결정한다.
④ 분할 그룹별로 분할된 검사점 저장 기법에 따라서 검사점 정보들을 저장한다.
⑤ 검사점 정보 저장이 완료되면 프로세스의 수행을 계속한다.
⑥ 프로세서의 수행이 종료될 때까지 검사점에 도달할 때마다 ②~⑤의 과정을 반복한다.

다중 분할된 검사점 저장 기법에서는 매 검사점마다 분할 그룹의 크기가 변경될 수 있으며, 이 때 네트워크 부하, 즉 사용 가능한 네트워크 대역폭의 크기와 변경하는 분할 그룹의 크기와는 비례 관계로 설정된다. 즉, 매번 검사점에서 사용 가능한 네트워크 대역폭을 검사하고, 사용 가능한 대역폭이 이전보다 줄어들었을 경우, 축소된 대역폭에 비례하여 분할 그룹의 크기를 줄인다. 또한 사용 가능한 대역폭이 증가되었을 때에는 그에 비례하여 분할 그룹의 크기를 늘릴 수 있다. 클러스터 응용 프로세스가 동작하는 중에 네트워크 부하는 초기치보다 높아질 수도 있고 낮아질 수도 있기 때문에 분할 그룹의 크기는 네트워크 부하가 증가할 경우, 초기값보다 하위의 크기로만 변경될 수 있도록 하고, 최대로는 초기 분할 그룹 크기까지 증가될 수 있도록 한다.

(그림 2)는 이와 같은 과정의 예를 나타내고 있다. (가)는 *T-1*과 *T*의 두 검사점 사이에서 네트워크 부하가 변화가 크지 않은 경우로 다음 검사점에서도 동일한 분할 그룹 크기



(가) 검사점에서 네트워크 부하의 변화가 없는 경우



(나) 검사점에서 네트워크 부하가 증가된 경우

(그림 2) 다중 분할된 검사점 저장 기법의 동작 과정

를 유지하면서 검사점 정보를 저장한다. (나)의 경우는 T 검사점에서의 네트워크 부하가 T-1 검사점보다 증가된 경우로, T-1 검사점에서의 분할 그룹의 크기는 4를 사용하였지만 T 검사점에서는 네트워크 부하의 증가로 인하여 한 단계 아래 그룹 크기 2를 사용하여 검사점 정보를 저장한다.

이와 같은 알고리즘은 매우 간단한 방법으로 구현할 수 있으며 O(1)의 복잡도를 가진다. 실제로 구현된 알고리즘을 통하여 매 검사점마다 동적인 그룹 분할을 수행하는 데에 걸리는 시간은 전체 검사점 수행 시간에 비하여 무시할 수 있을 정도이기 때문에 Libckpt[13]와 마찬가지로 시스템의 전체 성능에는 영향을 주지 않는다. 또한 매 검사점마다 네트워크 대역폭을 측정하는 과정도 본 알고리즘에서 수행하지 않고 클러스터를 구성하는 지능형 스위치의 기능에 의하여 직접 제공받기 때문에 별도의 추가적인 오버헤드는 발생되지 않는다.

4. 벤치마크 환경

다중 분할된 검사점 저장 기법과 기존의 분할된 검사점 저장 기법의 입출력 성능을 비교 평가하기 위해서는 다수의 노드에서 프로세스를 수행할 수 있는 클러스터 응용 프로그램을 실행하고, 검사점에서 각 노드들로부터 분산 RAID로 발생되는 입출력 부하를 분석하여야 한다. 클러스터 응용 프로그램으로는 TOP 500 컴퓨터를 선정하는 데에 사용하는 Linpack HPC(High-Performance Computing) 벤치마크를 사용하였다. 본 연구에서는 HPC는 Linux 기반으로 MPI (Message Passing Interface)를 이용하여 노드간 메시지 전송을 수행하는 버전을 사용하였으며 Libckpt 라이브러리를 이용한 프로그램 함수를 HPC 벤치마크 프로그램에 첨가함으로써 시스템에 추가적인 부하를 가하지 않고 검사점 정보를 저장하는 알고리즘을 구현하였다. 클러스터 시스템의 검사점에서의 입출력 성능은 메시지 기록에 소요되는 시간을 포함하여 검사점 정보를 저장하는 데 걸리는 시간, 즉 입출력 응답 시간(IO response time)과 입출력 처리량(Throughputs)을 통하여 직접적으로 나타낼 수 있다.

벤치마크를 실행시키기 위하 클러스터 시스템의 구성 환경은 <표 1>과 같다. 클러스터를 구성하는 노드로는 16대의 PC를 사용하였으며 HPC의 가상 프로세스 생성 기능을 통하여 최대 512개의 노드들이 가상적으로 동작되는 환경을 구현하였다. 이들 노드들은 100Mbps의 Ethernet 스위치 허브를 통하여 상호간에 연결된다. 각 노드는 RedHat Linux 6.2를 운영체제로 사용하고, 512MB의 메인 메모리와 300GB의 용량을 가지는 하나의 하드디스크를 포함하고 있으며 각 디스크 공간 중 100GB 영역을 클러스터 저장 장치 공간으로 할당하였다.

검사점 정보를 저장하는 전역 저장 장치는 기본적으로는 분산 RAID 구조를 지원하는 네트워크 저장 장치로 구성하였으며 총 1.6TB에 이르는 단일 입출력 주소 공간을 형성한다. 각 노드의 검사점 정보들은 디스크의 물리적 위치와 주

〈표 1〉 클러스터 시스템의 구성 환경

Parameters	Benchmark Environments
CPU	Intel Pentium4 2.8GHz
메모리 크기	512Mbyte SDRAM
L2 캐쉬 크기	1Mbyte
네트워크	100Mbps Intel Express Pro100
지역 디스크	300GB Seagate S-ATA On-board 100GB for Storage, 200GB for System
분할 그룹 크기	4, 16 (분할된 검사점 저장 기법) 1, 2, 4, 8, 16(다중 분할된 검사점 저장 기법)

소 배열에 관계 없이 매핑되어 단일 입출력 공간에 저장된다. 단일 입출력 공간은 최소 2개에서 최대 16개까지의 디스크들이 참여하여 형성할 수 있지만 본 연구에서는 각 노드에 1개씩 연결된 16개의 디스크들을 모두 사용하는 고정 크기의 단일 입출력 공간을 구성하였다. 분산 RAID는 Linux 운영체제에서 원격 디스크 제어 기능에 의하여 지원되므로 별도의 RAID 구성 프로그램을 사용하지는 않지만 전체 분산 RAID에 걸쳐서 단일한 주소 공간을 부여하기 위해서는 RPC(Remote Procedure Call) 기반의 원격 주소 공간 할당 프로세스가 사용된다[12].

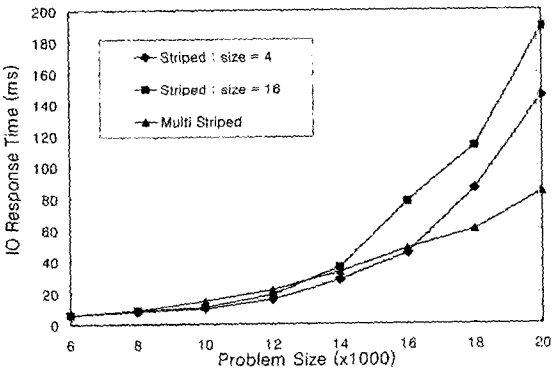
본 논문에서 제안된 다중 분할된 검사점 저장 기법과 기존의 분할된 검사점 저장 기법의 비교 평가를 위한 벤치마크에서 분할 그룹의 크기는 분할된 검사점 저장 기법의 경우 프로세스가 수행되기 이전에 결정되어야 한다. 반면에 다중 분할된 검사점 저장 기법은 프로세스 수행 중에도 매 검사점마다 동적으로 분할 그룹의 크기를 변경하기 때문에 분할 그룹의 크기를 미리 결정할 필요가 없다. 본 논문에서는 두 가지 검사점 저장 기법을 정량적으로 비교하기 위하여, 분할된 검사점 저장 기법에서는 4와 16의 두 가지 분할 그룹 크기에 대한 성능 평가를 수행하고 다중 분할된 검사점 기법의 성능 평가 결과와 비교함으로써 두 기법의 차이점을 분석하였다.

성능 평가를 위하여 사용된 Linpack HPC 벤치마크에는 100x100 크기와 1000x1000 크기의 행렬 연산을 수행하는 일련의 연산 문제들이 포함되어 있으며 Linpack의 핵심 내용인 LAPACK과 BLACS 라이브러리의 MPI-2 버전이 포함되어 있다. HPC는 문제의 크기를 변화시킴으로써 시스템 부하를 변화시킬 수 있다. 따라서 벤치마크에서는 문제 크기를 6000에서 20000까지 부가하여 시스템 부하를 변화시키고, 프로세스가 수행되면서 각 검사점에서 입출력 성능 평가 지수들을 측정함으로써 각 검사점 저장 기법의 입출력 성능을 평가하였다. 이를 위하여 HPC 실행 프로그램에 각 검사점에서의 검사점 정보를 저장하는 프로세스를 발생시키는 Libckpt 함수를 결합하여 주기적으로 검사점 정보를 저장하도록 프로그램을 수정하였다. 검사점 정보의 크기는 부가되는 부하의 크기에 비례하여 증가하므로 부하가 증가될수록 입출력 요구와 입출력 부하는 커지게 된다.

5. 성능 평가 결과 및 분석

(그림 3)은 클러스터 프로세스가 수행되는 동안에 모든 검사점에서의 검사점 정보에 대한 입출력 응답 시간(IO response time)의 평균값을 나타낸다. 분할된 검사점 저장 기법에서는 분할 집합의 크기와 관계 없이 프로세스의 브하가 증가됨에 따라서 입출력 응답 시간이 급격하게 증가되지만, 다중 분할된 검사점 저장 기법은 비교적 점진적으로 증가되고 있다. 이는 HPC 벤치마크, 즉 클러스터 프로세스가 수행되는 동안에 사용 가능한 네트워크 대역폭이 실제로 변화하고 있음을 나타낸다. 특히 프로세스의 부하가 낮을 경우에는 두 가지 검사점 저장 기법이 입출력 응답시간에 있어서 큰 차이를 보이지는 않지만, 부하가 일정 크기 이상으로 증가하면 네트워크 부하도 심하게 변화되면서 고정된 분할 그룹을 가지는 기법이 더 이상 최적이 아니라는 사실을 나타내 주고 있다. 그러나 분할 그룹의 크기가 가변적으로 조정되는 경우에는 각 검사점에서 최대의 병렬성을 제공할 수 있도록 분할 그룹의 크기를 조절하기 때문에 각 검사점 정보 저장에 소요되는 시간이 상대적으로 줄어들게 된다.

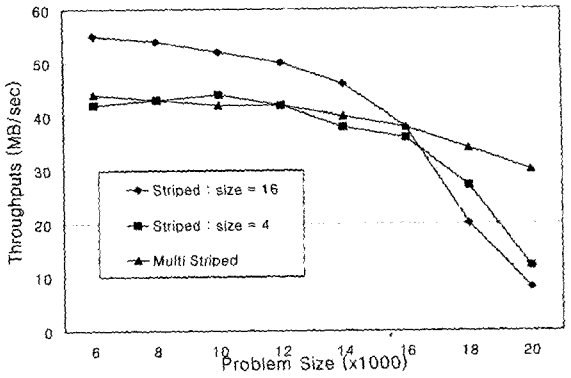
<표 2>는 이와 같은 변화의 분포를 잘 나타내 주고 있다. 표에서 나타내는 값은 HPC 벤치마크 수행 중에 해당 부하에서 네트워크 가용 대역폭에 따라서 선택되는 분할 그룹 크기의 분포를 나타낸다. 이는 부하가 증가되면 네트워크 대역폭도 줄어들면서, 상대적으로 작은 크기의 분할 그룹이 선택되는 경우가 많아지고, 부하가 작으면 네트워크 가용 대역폭이 증가되어 보다 병렬성이 큰 분할 그룹 크기



(그림 3) 검사점 정보에 대한 평균 입출력 응답 시간
(Average IO Response Time)

〈표 2〉 부하에 따른 분할 그룹의 크기 분포(%)

Group Size / Problem Size	16	8	4	2	1
7500	49.0	47.6	2.3	1.1	0.0
10000	25.9	52.3	19.3	2.1	0.4
12500	4.7	41.0	33.3	17.9	3.1
15000	0.8	12.1	47.3	22.6	17.2
17500	0.9	7.1	21.4	36.3	34.3
20000	0.0	0.0	0.5	13.3	86.2



(그림 4) 검사점 정보에 대한 처리량*(Throughputs)*

가 선택될 수 있는 경우가 많아진다는 결과를 나타내고 있다. 이러한 결과는 네트워크 부하와 시스템 부하에 따라서 적절한 크기의 병렬성을 제공하는 분할 그룹의 크기를 결정하는 것이 검사점 정보 저장 성능과 밀접한 관계가 있음을 나타낸다.

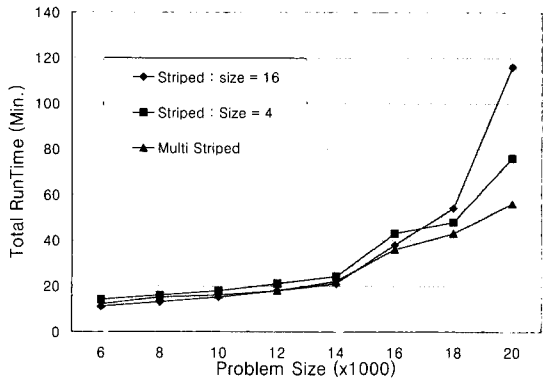
(그림 4)는 클러스터 프로세스가 수행되는 동안 모든 검사점에서 검사점 정보의 전송 능력을 나타낼 수 있는 입출력 처리량(Throughputs)을 나타낸다. 검사점 정보에 대한 입출력 처리량은 입출력 응답 시간과 밀접한 관계가 있다. 모든 노드들이 동일한 특성을 가지는 클러스터의 경우, 각 노드의 검사점 정보의 크기는 거의 동일하기 때문에 입출력 응답 시간이 짧을수록 단위 시간당 데이터 전송 능력은 증가된다. 따라서 다중 분할된 검사점 저장 기법이 상대적으로 높은 입출력 처리량을 나타낸다. 그러나 기존의 분할된 검사점 저장 기법은 높은 부하에서 처리량이 급격히 감소된다.

(그림 5)는 HPC 벤치마크 프로그램의 전체 수행 시간을 나타내고 있다. 프로그램의 전체 수행 시간은 노드에서의 프로세스 수행 시간들의 합과 검사점 정보 저장 시간들의 합으로 다음과 같이 나타낼 수 있다.

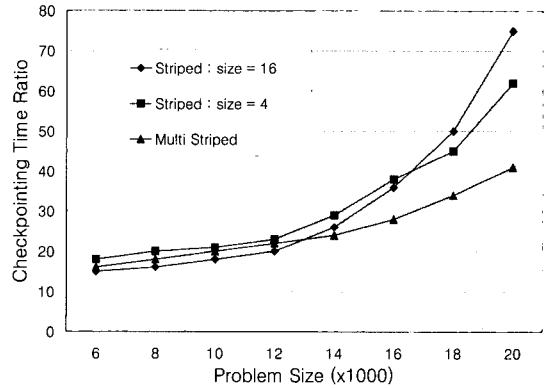
$$Total\_RunTime = Total\_Process\_RunTime + Total\_Checkpoint\_Time$$
$$Total\_Process\_RunTime = NP_1 + NP_2 + \ldots + NP_{k-1} + NP_k$$
$$Total\_Checkpoint\_Time = CP_1 + CP_2 + \ldots + CP_{k-1} + CP_k$$

NP_t : 검사점 t직전의 프로세스 수행 시간
CP_t : 검사점 t에서의 검사점 정보 저장 시간
k : 총 검사점의 수

동일한 벤치마크 프로그램이 실행되는 경우, 각 노드에서의 프로세스 수행 시간들NP_k는 동일하기 때문에 수행 시간의 차이는 검사점 정보 저장 시간 CP_k 에 의하여 결정된다. 분할된 검사점 저장 기법의 경우에는 병렬 입출력이 수행될 때 네트워크의 부하에 따라서 대기 시간이 지연되거나 바로 처리되기도 한다. 반면에 다중 분할된 검사점 저장 기법에서는 병렬 입출력이 즉시 처리 가능하도록 분할 그룹의 크기를 나누지만, 순차 저장 횟수가 증가되기 때문에 전체적인 검사점 정보 저장 시간이 증가될 수 있다. (그림 5)의 벤



(그림 5) HPC 벤치마크 실행 시간(*Total_RunTime*)



(그림 6) 프로세스 수행 시간 대비 검사점 정보 저장 시간 (*Ratio_Checkpoint*)

치마크 결과는 부하가 증가됨에 따라서 두 기법간의 전체 수행 시간의 차이가 점차로 증가되고, 최대 부하에 접근하면 급격한 차이를 보인다. 따라서 본 연구와 같은 클러스터 환경에서는 부하가 증가할수록 병렬 입출력 요구에 대한 네트워크 지연 시간이 순차 저장 때문에 증가되는 시간보다 커진다는 사실을 알 수 있다. 이러한 네트워크 지연 시간이 검사점 정보 저장 시간을 대부분 차지하는 경우, 부하가 증가할수록 전체 프로세스 수행 시간 중에서 검사점 저장에 소요되는 시간은 증가된다.

(그림 6)은 전체 프로세스 수행 시간 중 검사점 저장에 소요된 시간의 비율 *Ratio_Checkpoint*을 나타내고 있다. 여기서 *Ratio_Checkpoint*는 다음과 같이 계산될 수 있다.

$$Ratio\_Checkpoint = \frac{Total\_Checkpoint\_Time}{Total\_Run\,Time}$$

어느 기법에서나 디스크 장치 수준에서의 입출력 대기 시간(IO request waiting time)은 거의 동일하므로 이러한 차이는 네트워크 지연에 의한 차이라고 할 수 있으며, 사용 가능한 네트워크 대역폭 내에서 입출력을 수행하면 그렇지 않은 경우에 비하여 네트워크 지연을 줄일 수 있음을 알 수 있다.

6. 결론 및 향후 연구

다중 분할된 검사점 저장 기법은 클러스터 시스템의 네트워크 부하 상태에 따라서 각 검사점에서 저장하는 검사점 정보를 그 순간의 네트워크 대역폭 이내로 전송할 수 있는 분할 그룹 크기를 사용하여 저장할 수 있으므로 프로세스가 수행되는 동안 네트워크 트래픽이 동적으로 변화하는 클러스터 시스템에서도 최적의 입출력 성능을 나타낼 수 있다. 검사점에서 보다 우수한 입출력 성능을 얻는 것은 곧 전체 프로세스의 수행 시간을 줄일 수 있다는 것을 의미하므로 클러스터 시스템의 결함 허용 알고리즘을 동작시키는 데에 소요되는 오버헤드를 줄일 수 있고, 결과적으로 프로세스의 수행 성능을 증대시킬 수 있다. 성능 평가 결과, 클러스터 시스템의 노드가 증가할수록, 고정된 분할 그룹 크기를 가지는 검사점 저장 기법보다는 검사점마다 동적으로 분할 그룹 크기를 조절하는 검사점 저장 기법이 입출력 응답 속도, 최대처리량 등의 주요 성능 평가 요소들에 대하여 우수한 성능을 나타내고 있다. 특히 시스템의 부하가 증가할수록, 그리고 클러스터 시스템을 구성하는 노드의 수가 많아질수록 뚜렷한 성능의 차이를 보인다. 따라서 부하가 높은 프로세스를 처리하는 큰 규모의 클러스터 시스템에서는 다중 분할된 검사점 저장 기법을 사용하여 시스템의 결함 허용 성능을 증대시킬 수 있다. 또한 이와 같은 결과는 차후의 대규모 클러스터 시스템을 설계할 때에 필수적으로 고려되어야 하는 부분으로 고가용성, 고성능, 고 신뢰도를 가지는 시스템의 설계에 많은 도움을 줄 수 있다.

다중 분할된 검사점 저장 기법에 대한 성능 평가의 결과는 클러스터 시스템의 네트워크 부하는 단지 각 노드로 분산된 프로세스들 간의 통신만으로 이루어진 것이 아니라 외부 네트워크와 각 노드의 운영체제, 그리고 시스템의 여러 서브 프로세스들이 발생시키는 통신 부하들도 포함되어 있으며, 이러한 통신 부하들로 인하여 클러스터 응용 프로세스들이 사용 가능한 네트워크 대역폭이 큰 폭으로 변한다는 것을 시사하고 있다. 따라서 지금까지의 연구 결과들과는 달리, 클러스터 시스템에서 병렬 프로세스들이 사용 가능한 최대 네트워크 대역폭은 고정적인 관점이 아닌 동적인 관점에서 접근해야만 한다. 따라서 클러스터 노드들에 대한 부하 분배 문제나 분산 RAID에 대한 병렬 입출력을 위한 네트워크 대역폭, 그리고 결함 허용을 지원하는 데에 사용되는 여러 가지 자원의 분배 문제는 모두 동적인 요소를 포함하여야 하며, 향후 연구에서는 프로세서나 네트워크 자원에 대한 동적 할당의 관점에서 문제를 접근할 필요가 있다.

참 고 문 헌

[1] L. Alvisi, B. Hoppe and K. Marzullo, "Nonblocking and Orphan-Free Message Logging Protocols," In Proceedings of the 23th Symposium on Fault-Tolerant Computing, pp. 145-154, 1993.

[2] P. J. Braam et al., "The Lustre Storage Architecture," Cluster File System. Inc., Mar., 2003.
[3] G. Cao and M. Singhal, "On Coordinated Checkpointing in Distributed Systems," IEEE Transactions on Parallel and Distributed Systems, Vol.9, No.12, 1998.
[4] Y. Chang et al, "Performance Evaluation of the Striped Checkpointing Algorithm on the Distributed RAID for Cluster Computer," Lecture Notes in Computer Science, Vol. 2658, pp.955-962, 2003.
[5] P. Carns et al. "PVFS: A Parallel File System for Linux Clusters." In Proceedings of the 4th Annual Linux Showcase and Conference, pp.317-327, 2000.
[6] E. Elnozahy and W. Zwaenepoel, "On the Use and Implementation of Message Logging," In Proceedings of 24th International Symposium on Fault-Tolerant Computing, 1994.
[7] E. N. Elnozahy, and W. Zwaenepoel, "Manetho: Transparent Rollback-Recovery with Low Overhead, Limited Rollback and Fast Output Commit," IEEE Transactions on Computers, Vol.41 No.3, pp.526-531, 1992.
[8] J. H. Hartman et al., "The Zebra Striped Network File System," ACM Transactions on Computer System, Vol. 13 No.3, pp.274-310, 1995.
[9] K. Hwang and Z. Xu, "Scalable Parallel Computing". McGraw-Hill, 2000.
[10] K. Hwang, H. Jin, R. Ho and W. Ro, "Reliable Cluster Computing with a New Checkpointing RAID-x Architecture," Proceedings of 9-th Workshop on Heterogeneous Computing, Cancum, Mexico, 2000.
[11] K Hwang, H. Jin, and R. Ho, "RAID-x: A New Distributed Disk Array for I/O Centric Cluster Computing," Proceedings of 9th High-Performance Distributed Computing Symposium, Pittsburgh, 2000.
[12] K. Hwang, H. Jin, E. Chow, C. Wang, and Z. Xu, "Designing SSI Clusters with Hierarchical Checkpointing and Single IO Space," IEEE Concurrency Magazine, 1999.
[13] J. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent Checkpointing Under UNIX," In Proceedings of USENIX Winter 1995 Technical Conference, 1995.
[14] J. Plank K. Li, and M. Puening, "Diskless Checkpointing," IEEE Transactions on parallel and Distributed Systems, 1998.
[15] K. W. Preslan et al., "A 64bit, Shared Disk File System for Linux," In Proceedings of the 16th IEEE Mass Storage Systems Symposium, pp.22-41, 1999.
[16] R. Sandberg, "The SUN Network Filesystem: Design, Implementation and Experience," SUN Microsystems, Inc., pp.119-130, 1985.
[17] F. Schmuck et al., "GPFS: A Shared-Disk File System for Large Computing Clusters," In Proceedings of the FAST Conference on File and Storage Technologies, pp.231-234, 2002.
[18] N. Vaidya, "Staggered Consistent Checkpointing," IEEE Transactions on Parallel and Distributed Systems, Vol.10, No.7, 1999.

장 윤 석

e-mail : cosmos@sparc.snu.ac.kr
1988년 서울대학교 물리학과(이학사)
1990년 서울대학교 컴퓨터공학과(공학석사)
1998년 서울대학교 컴퓨터공학과(공학박사)
1994년~2002년 대진대학교 컴퓨터공학과 전임강사 및 조교수
2003년~현재 대진대학교 컴퓨터공학과 부교수
2000년~2001년 Visiting Scholar in Dept. of EE-Systems, University of Southern California
관심분야 : 컴퓨터구조, 클러스터 컴퓨터, RAID, 전자상거래 구조 등