2000년 한국정보처리학회 추계 학술발표논문집 제 7 권 제 2 호

이기종 클러스터의 JVM 기반 단일시스템이미지 구현

한동헌*, 김양우**
동국대학교 정보통신공학과
e-mail : *dhan73@hanmail.net, **ywkim@dgu.edu

A JVM based Implementation of a Single System Image for Heterogeneous Clusters

Dong-Hun Han*, Yang-Woo Kim**
Dept. of Information & Communication, Dong-Guk University

요 약

지난 10년간 인터넷은 정보화 시대의 새로운 패러다임으로 사회 전반을 변화시켜 왔다. 네트워크 요청이 크게 증가하고 대량의 정보처리가 요구되면서 이 기간 동안 고성능 컴퓨팅 환경에 대한 연구가 활발하게 진행되었다. 낮은 가격과 높은 가용성을 유지하면서 고성능 컴퓨팅 환경을 제공하려는 노력은 클러스터링 기술을 발전시켰지만, 기존 클러스터 시스템들은 운영체제 및 하드웨어에 종속적이기 때문에 높은 확장성을 제공하지 못하였다. 또한 여러 대의 시스템이 연결되는 클러스터 시스템은 각각의 운영체제를 가지고 있지만, 시스템 전체 자원을 효율적으로 이용하기 위해서는 하나의 시스템 이미지를 가지고 작업을 처리해주는 단일시스템이미지(SSI) 제공이 필요하다.

기존의 SSI 제공 기술을 JVM(Java Virtual Machine) 계층을 기준으로 살펴보면, JVM 상위계층에서 구현된 SSI는 불완전하고, JVM 하위계층에서 구현된 SSI는 Java 자체의 성능 향상 기능을 활용하지 못하는 단점을 가진다. 이러한 단점들을 해결한 JVM 계층에서의 SSI가 구현되고 현재 연구가 진행중이지만, 이 역시 동기종 클러스터 시스템에 국한되어 있다.

본 논문에서는 클러스터 시스템의 SSI 구현 방법들을 각 구현 계층별로 비교하여 살펴보고, JVM을 기반으로 SSI를 제공함으로써 이기종 클러스터 시스템에까지 SSI를 확장, 제공하는 방안을 모색하고자 한다.

1. 서론

미션크리티컬한 컴퓨팅 환경이 그 동안 분산 네트워크를 기반으로 한 고성능 컴퓨팅 환경으로 변화, 발전하면서, 고가용성과 확장성에 대한 요구가 한층 증가하여 왔다. 이러한 요구의 최상의 해결책으로 다시 주목받기 시작한 것은 1986년 Digital사의 VAX 시스템을 통해 처음 소개되었던 클러스터링 기술이다. 클러스터링은 여러 대의 시스템을 서로 연결하여 마치 하나의 커다란 단일시스템처럼 사용할 수 있다. 따라서 단일컴퓨터 시스템의 처리능력 한계를 극복하면서, 시스템의 가용성을 높이기 위해, 고가용성과 확장성을 동시에 추구할 수 있다.

그러나, 기존 클러스터 시스템들은 운영체제 및 하드웨어에 종속적인 클러스터링 기술을 사용하기 때문에 높은 확장성을 제대로 제공하지 못하였다. 또한 클러스터 시스템의 각 노드는 각각의 운영체제를 가지고 있기 때문에, 시스템 전체 자원을 효율적으로 이용하여 원하는 작업을 분산 처리한다는 것은 엄청난 비용손실과 장애발생 가능성의 증가를 초래한다. 이러한 단점을 해결하기 위해 최근 등장한 단일시스템이미지(SSI : Single System Image)는 여러 대의 시스템이 연결되더라도 하나의 시스템 이미지를 가지고 작업을 처리함으로서 시스템 전체 성능을 향상시켜준다[1].

SSI 구현 방법에는 JVM(Java Virtual Machine) 계층을 기준으로 보았을 때, [그림2]와 같이 세 가지 접근 방

법이 있다. SSI를 JVM 상위계층에서 소프트웨어적으로 구현하면 완전하게 SSI를 제공할 수 없고, JVM 하위계층의 하부구조에서 구현하면 멀티쓰레드 등 Java 자체의 성능 향상 기능을 활용하지 못한다. 이러한 단점을 보완하여 JVM 계층에서 SSI를 제공하는 연구가 진행되고 있으나, 이 역시 동기종 클러스터 시스템에 국한되어 있다.

본 논문에서는 클러스터 시스템의 SSI 구현 방법들을 각 구현 계층별로 비교하여 살펴보고 JVM을 기반으로 이기종 클러스터 시스템에까지 SSI를 확장, 제공하는 방안을 모색하고자 한다.

앞으로 본 논문의 2장에서 SSI의 정의와 제공 방법들을 시스템 레벨에서 살펴보고, 3장에서 본 논문에서 처리하려는 작업 어플리케이션을 정의한다. 4장에서는 JVM을 기준으로 SSI 구현 방법에 대하여 구현 계층별로 비교하여 살펴보고, 5장에서 JVM을 기반으로 이기종 클러스터 시스템에서 SSI를 구현하는 방법을 설명한다. 마지막으로, 6장에서 결론과 향후연구방향에 대해 기술한다.

2. 단일시스템이미지 정의

SSI는 단순히 메모리 상에 존재하는 운영체제의 이미지가 하나라는 것을 의미하지는 않는다. 클러스터 시스템에서 제공되는 SSI는 다음과 같은 특성을 가져야 한다[1].

- 단일 시스템(Single System) : 전체 시스템은 사용자에 의해 하나의 시스템으로 보인다.
- 단일 제어(Single Control) : 최종 사용자나 시스템 관리자는 단일 인터페이스를 가진 한 지점에서 클러스터 서비스들을 최적화시킬 수 있다.
- 대칭성(Symmetry) : 사용자는 임의의 어떤 노드에서도 동일한 클러스터 서비스를 사용할 수 있다.
- 위치 투명성(Location Transparent) : 사용자는 임의의 어떤 서비스를 제공하는 물리적 장치가 실제 어디에 위치하는지에 대해 알 필요가 없다.

SSI는 클러스터 시스템에서 시스템 레벨별로 각각 제공될 수 있는데, 한 레벨에서의 제공 방법은 다른 레벨의 제공 방법에 부분적으로 겹쳐지기도 한다[1].

2.1 어플리케이션 소프트웨어 레벨

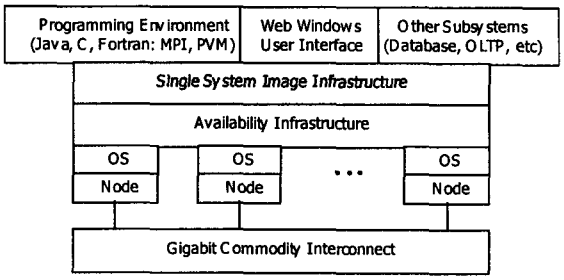
웹 브라우저나 다양한 병렬 데이터베이스를 예로 들 수 있다. 사용자는 어플리케이션을 통해 SSI를 제공받으며 사용하는 클러스터 시스템이 어떠한 구조를 가지는지 알 필요가 없다. 이와 같은 제공 방법은 워크스테이션이나 클러스터를 위한 SMP 어플리케이션의 수정을 요구한다.

2.2 하드웨어 혹은 커널 레벨

가장 이상적인 방법으로 SSI는 운영체제나 하드웨어에 의해 제공된다. 그러나, 실제로 구현하기가 어렵고, 특히 이기종 클러스터에서 SSI를 제공한다는 것은 극히 어렵다. 시스템 내의 대부분의 하드웨어와 운영체제를 독자적으로 구성하고 있는 밴더에 의해서만 제공되고 있다.

2.3 커널 상위레벨

대부분의 가능한 제공 방법으로 [그림1]과 같이 커널 상위레벨에서 SSI를 제공한다. 이 방법은 플랫폼 독립적이기 때문에 어플리케이션의 수정을 요구하지 않는다. 많은 클러스터 작업 관리 시스템들(Job Management Systems)이 이미 이 제공 방법을 적용하고 있다.



[그림1] 커널 상위레벨에서의 SSI 제공

3. 어플리케이션 정의

본 논문에서 적용하는 어플리케이션은 Java Server Applications(JSAs)으로서 2-tier 어플리케이션이다. JSA는 클라이언트로부터 요청(request)을 순차적으로 받아들이고 그 요청들을 처리하기 위해 외부 데이터베이스에 접근하며, 요청들은 상호작용을 하기도 한다.

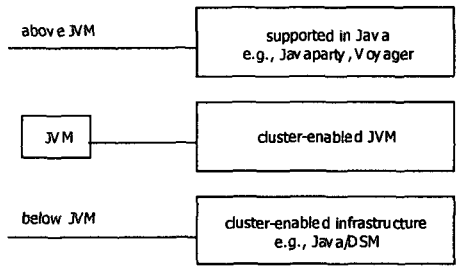
본 논문의 목적이 주어진 작업에 기종에 상관없이 더 많은 하드웨어 요소를 추가함으로써 어플리케이션의 수용력을 확장하는데 있기 때문에 본 논문에서 고려한 어플리케이션 특성은 다음과 같다.

- 높은 수용력(High Capacity) : 단위 시간당 사용자 Bytecodes 의 수
- 높은 처리능력(High Throughout) : 단위 시간당 요청 수
- 공정성(Fairness) : 각 요청들 간의 공정성
- 지속적인 실행(Long Running) : 오류에 대한 탄력성
- 보안(Security) : 기존 JVM에 대해 우수한 보안능력
- 투명성(Transparency) : JVM 상에서 JSA가 실행되기 위한 코드 수정의 불필요
- 공간, 시간적 처리능력(Over space and time) : 여러 클라이언트로부터 동시에 발생하는 요청들과 동일 혹은 다른 클라이언트로부터 발생하는 순차적인 요청들의 처리
- 공유(Sharing) : 데이터베이스에 의해 공유되는 지속적인(persistent) 데이터와 Java 개체에

의해 공유되는 일시적인(transient) 데이터 혹은 데이터베이스로부터 캐쉬한 데이터의 처리

4. 단일시스템이미지 구현 방법

본 논문에서는 JVM을 기반으로 JSA을 처리하기 위해 제공되어야 하는 SSI를 구현한다. SSI를 구현하는 방법을 JVM 계층을 기준으로 분류하여 보면 [그림2]와 같다.



[그림2] 단일시스템이미지 구현 방법

4.1 JVM 상위계층

멀티쓰레드로 구성된 Java 어플리케이션조차도 클러스터 시스템 상에서는 한 노드에서 수행되는데, 이를 분산, 병렬 처리하기 위해서는 프로그래머가 어플리케이션의 분산 특성을 명시적으로 처리해야 한다. 이 때 주로 Java RMI, CORBA와 같은 표준 통신 라이브러리를 이용한다.

Javaparty는 객체들을 원격 객체로서 투명하게 재기술함으로서 Java 어플리케이션을 분산 처리하는 접근방법을 이용했고[2], Voyager는 Java RMI를 이용하여 원격 Java 클래스를 지원하였다[3].

이와 같이 JVM 상위계층에서 소프트웨어적으로 SSI를 구현하면 Java 어플리케이션의 분산 복잡성을 완전하게 해결하지 못하고 어플리케이션의 분산된 특성이 프로그램으로부터 완전하게 숨겨지지 않는다. 즉 SSI가 불완전하게 되며[4][5], 처리속도도 느려진다.

4.2 JVM 하위계층

분산 공유 메모리 등 JVM 하위계층의 하드웨어나 운영체제 상에서 SSI를 구현하면 처리속도를 향상시킬 수 있다. 그러나, 멀티쓰레드 등 Java 자체의 성능 향상 기능을 완전하게 활용하지 못하고[6], 이기종간의 클러스터 자체가 어려워진다. 이 방법은 가장 이상적인 방법이지만 실제 구현하기가 가장 어렵다.

그러나, Java 기술을 적용한 Java/DSM은 JVM의 힙(heap)을 TreadMarks의 분산 공유 메모리 시스템 상에서 구현하면서 이기종 클러스터 시스템에서 SSI를 구현하는 방법을 제시하였다[7]. 또한 JESSICA는 멀티쓰레드로 구성된 Java 어플리케이션의 분산, 병렬 처리를 제공하기 위해 표준 UNIX 운영체제 상에서 운영되는 미들웨어로 전역 객체 공유를 분산 공유 메모리 구조를 통해 구현하였다[8].

4.3 JVM 계층

JVM 상하위계층에서 SSI를 구현하면서 발견된 문제들을 보완하여 JVM 계층에서 SSI를 구현하는 연구가 진행 중이지만, 현재 동기종 클러스터 시스템에 국한되어 있다.

예를 들어, cJVM은 클러스터의 노드들 상에서 분산된 형태로 수행되는 기존의 JVM에 SSI를 제공하는 새로운 JVM 모델을 제시하였다. 하지만 이 역시 이기종 클러스터 시스템에는 적용되지 못하고 있다[9][10].

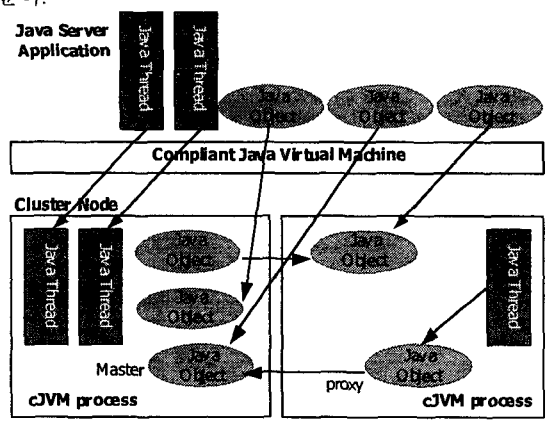
5. JVM 기반 단일시스템이미지 구현 모델

JVM은 어플리케이션 프로그램들이 시스템 플랫폼에 독립적으로 수행될 수 있도록 하는 가상머신이기 때문에[11], JVM을 기반으로 해서 SSI를 구현하면 이기종 클러스터 시스템을 구성하여 프로그램의 수정없이 곧바로 실행할 수 있다.

또한, JVM 기반 SSI는 어플리케이션의 작업을 클러스터 시스템의 컴퓨팅 자원에 분산시킴으로써 어플리케이션을 위한 시스템의 확장성뿐만 아니라 어플리케이션 자체의 확장성도 향상시킨다.

5.1 JVM 기반 단일시스템이미지 구현 방법

[그림3]에서 상위부분은 프로그래머에 의해 보여지는 Java 어플리케이션의 쓰레드와 객체들을 보여주고 있으며, 하위부분은 JVM에 의해 클러스터의 각 노드에 걸쳐 쓰레드와 객체들이 분산되어있는 것을 보여준다.



[그림3] Cluster JVM

여기서는 JVM 계층에서 SSI를 제공하기 위해 새로운 객체 모델을 고려하였다. 각 노드에 분산된 객체들에게 접근하기 위해서는 [그림3]과 같이 master-proxy 모델을 사용한다. 객체가 생성된 노드에 그 객체의 master가 있고, 이 객체에 접근하기 위해 다른 노드에서는 proxies가 사용된다. master 객체는 프로그래머에 의해 정의되며 proxy는 원격 객체에 접근하기 위해 사용되는 대리자 역할을 하게 된다.

어플리케이션이 master 객체를 사용하는지 proxy를 사용하는지 알 필요가 없도록 하기 위해서 proxy 객

소드 테이블 등과 같은 내부 표현을 가지고 구현된다. 따라서 한 객체의 모든 proxies 들은 단일 클래스 객체 하에서 공존하게 된다.

또한 Java 구조(semantics)에 위배되지 않으면서 효율적인 proxy를 구현하기 위해서 클래스들이 로딩되어 있는 동안 모든 클래스들을 분석하고 그것들이 힙(heap)에 접근하는 특성을 고려하여 메소드를 분류한다. 이 정보를 이용하여 각 메소드에 대해 가장 효율적인 proxy를 구현하게 된다.

5.2 JVM 기반 단일시스템이미지 구현 모델의 장점

JVM 기반 단일시스템이미지 구현 모델은 객체와 쓰레드의 실제 운영을 어플리케이션에게 투명하게 보여주면서 SSI를 제공한다. 따라서 수많은 쓰레드와 객체들로 구성된 어플리케이션이 JVM 상에서 수행되어질 때, 어플리케이션은 쓰레드들이 어디에서 수행되고 객체들이 실제로 어디에 존재하는지 알 필요가 없게 된다.

또한 객체 접근에 대한 지역성(locality)을 향상시키기 위해 객체 복제를 사용하는 것 등을 고려하여 각 객체들을 구성함으로써 성능 향상을 얻을 수 있으며, JVM을 통해 여러 노드에 쓰레드와 객체들을 분산시킴으로써 높은 확장성을 획득할 수 있다.

JVM의 분산 구현은 JVM의 원래 구조(semantics)를 유지하기 때문에 이기종 클러스터 시스템에서조차 어떤 프로그램도 수정 없이 안전하게 어플리케이션을 실행시킬 수 있고, 클러스터 시스템의 전체 자원의 활용을 가능하게 해준다.

6. 결론 및 향후연구방향

기존의 클러스터링 기술들은 동일 운영체제 및 하드웨어를 기반으로 한 동기종 클러스터 환경에 연구가 집중되어 왔다. 본 논문에서는 어플리케이션 프로그램들이 시스템 플랫폼에 독립적으로 수행될 수 있도록 하는 JVM을 기반으로 SSI를 구현하여 클러스터의 SSI 제공과 확장성 문제를 해결하고, 시스템을 효율적으로 구성하기 위한 방법을 모색하였다.

어플리케이션은 코드 수정 없이 이기종 클러스터 시스템에까지 확장, 제공될 수 있으며, 이러한 사실은 기존의 우수한 각 시스템들을 기종에 상관없이 클러스터 시스템으로 통합시키면서 대규모 어플리케이션을 처리할 수 있다는 것을 의미한다. 또한 저가의 PC를 이용하여 고가의 고성능 클러스터 시스템을 구성할 수 있게 해준다.

본 연구와 관련되어 앞으로 객체들이 모든 쓰레드들에 의해 공유될 수 있도록 분산 공유 메모리, 객체 적재(Shipping) 모델, 기능 적재(Function Shipping) 모델, 혼합 모델 등 JVM 기반 클러스터 시스템에 가장 적절한 메모리 모델 패러다임에 대한 연구가 필요하다. 또한, 이러한 메모리 모델을 결정하는 데에 분산 가비지 컬렉션(GarbageCollection), 객체 클러스터링, 마이그레이션(Migration), 캐쉉, 복제 등이 이슈가 될 수 있다.

한편 보다 높은 확장성을 얻기 위해서는 JVM 기반 클러스터 시스템의 최적화에 대한 연구가 필요하다. 여기서 최적화의 주된 초점은 데이터의 지역성을 향상시킴으로써 클러스터의 노드들 간의 통신량을 줄이는 데에 맞추어질 수 있다.

그리고 Java RMI, CORBA 등과 같은 소프트웨어적인 SSI 구현 방법과 Java/DSM 등과 같이 이미 이기종 클러스터의 SSI를 하드웨어적으로 구현한 방법들을 JVM 기반 SSI와 통합시키는 연구는 최적의 클러스터 시스템을 구성하는 데에 기여할 수 있을 것이다.

이뿐만 아니라, 급속히 발전하고 있는 인터넷, 네트워크 기술과 함께, JVM을 기반으로 한 이기종 클러스터링 기술은 그 범위를 확대하여 인터넷을 기반으로 하는 이기종 시스템 간의 고성능 분산 컴퓨팅(HPDC : High Performance Distributed Computing) 분야에까지 적용될 수 있어 이 분야에 많은 기여를 할 수 있을 것으로 기대된다.

참고문헌

[1] Kai Hwang, Zhiwei Xu, "**Scalable Parallel Computing: Technology, Architecture, Programming**", McGraw Hill, 2000

[2] M. Philippsen and M. Zenger, "**Javaparty: Transparent remote objects in Java**", Concurrency: Practice and Experience, 11(9): 1125-1242, 1997

[3] http://www.objectspace.com/ : voyager

[4] P. Launy and J. Pazat, "**A framework for parallel programming in Java**", Technical Report 1154, IRISA, December 1997

[5] D. Caromel and J. Vayssiere, "**A Java framework for seamless sequential, multi-threaded, and distributed programming**", In ACM 1998 Workshop on Java for High-Performance Network Computing, INRIA Sophia Antipolis, France, 1998

[6] James Gosling, "**The Java Language Specification, Second Edition**", chapter17, Addison Wesley Longman, Inc., June 2000

[7] A. Yu and W. Cox, "**Java/DSM a platform for heterogeneous computing**", In ACM 1997 Workshop on Java for Science and Engineering Computation, June, 1997

[8] M.J.M. Ma, C.L. Wang, F.C.M. Lau, and Z. Xu, "**JESSICA: Java-Enabled Single-System-Image Computing Architecture**" International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), June 28 - July 1, 1999, Las Vegas, Nevada, USA, Pages 2781-2787

[9] Y. Aridor, M. Factor, and A. Teperman, "**cJVM: A Single System Image of a JVM on a Cluster**", 1999 International Conference on Parallel Processing, September 21 - 24, 1999, Wakamatsu, Japan

[10] Y. Aridor, M. Factor, A. Teperman, T. Eilam and A. Schuster, "**A high performance cluster JVM presenting a pure single system image**", Java Grande Conference, Proceedings of the ACM 2000 conference on Java Grande, June 3 - 4, 2000, San Francisco, CA USA, Pages 168-177

[11] *Tim Lindholm and Frank Yellin*, "***The Java Virtual Machine Specification, Second Edition***", Addison Wesley Longman, Inc., March 1999