리눅스 클러스터의 고가용성 보장을 위한 커널 수준 그룹 통신 시스템

(A Kernel-Level Group Communication System for Highly Available Linux Cluster)

이 상 균 † 박 성 용 ††
(Sangkyoon Lee) (Sungyong park)

요 약 클러스터에 대한 관심이 증가함에 따라 클러스터의 고가용성(high availability)을 보장하기 위한 그룹 통신 시스템의 연구가 활발하다. 하지만 클러스터를 이용한 커널 수준 응용 프로그램 개발에 활용할 수 있는 커널 수준의 그룹 통신 시스템은 전무하며, 기존에 개발되어 있는 사용자 수준 그룹 통신 시스템을 그대로 커널 수준에서 사용하는 것은 무리가 있다. 본 논문에서는 클러스터의 고가용성 보장을 위하여 리눅스 커널 상에서 구현된 그룹 통신 시스템인 KCGCS(Kernel-level Cluster Group Communication System)를 설계하고 구현하였다. KCGCS는 기존 사용자 수준의 시스템들과는 달리 경량(light weight)의 핫빗(heartbeat) 메시지와 링(ring) 구조의 핫빗 메커니즘을 사용하여 확장성 있는 장애 상황 발견 메커니즘을 구현하였다. 또한 멤버쉽 관리 측면에서 KCGCS는 중앙 집중된 기존의 방법과는 달리 분산된 코디네이터를 사용하여 신뢰성을 향상시켰다.

키워드 : 그룹 통신, 리눅스 클러스터, 고 가용성, 리눅스 커널

Abstract With the increase of interests in cluster, there have been a number of research efforts to address the high availability issues on cluster. However, there are no kernel-level group communication systems to support the development of kernel-level application programs and it is not easy to use traditional user-level group communication systems for the kernel-level applications. This paper presents the design and implementation issues of KCGCS(Kernel-level Cluster Group Communication System), which is a kernel-level group communication module for linux cluster. Unlike traditional user-level group communication systems, the KCGCS uses light-weight heartbeat messages and a ring-based heartbeat mechanism, which allows users to implement scalable failure detection mechanisms. Moreover, the KCGCS improves the reliability by using distributed coordinators to maintain membership information.

Key words : Group Communication, Linux Cluster, High Availability, Linux Kernel

1. 서 론

1990년대 초중반 저렴한 가격과 높은 성능을 동시에 갖춘 범용 통신 장치들이 개발됨으로서 MPP(Massively Parallel Processor) 등 주류를 이루던 슈퍼컴퓨팅 장치들에 비해 상대적으로 저가인 워크스테이션이나 개인용 컴퓨터를 이용하여 제작된 클러스터가 비중 있게 슈퍼컴퓨터 시장에 등장하게 되었다. 이러한 클러스터의 출현과 더불어 클러스터에서 싱글 시스템 이미지(single system image)를 제공하기 위한 시스템 프로그램(예: 파일 시스템)이나 또는 사용자 수준의 클러스터 응용 프로그램들이 개발되기 시작하였고, 이러한 프로그램들은 고성능 또는 고가용성(high availability)을 달성하기 위하여 작성되었다.

한편, 커널 수준의 클러스터 소프트웨어는 유저 수준과의 경계 교차 오버헤드를 피할 수 있어 통신을 고속화 할 수 있으며, 시스템 자원에 대한 직접적 접근이 가능하기 때문에 사용자 수준에 비하여 상대적으로 효율적인 프로그래밍이 가능하고, 클러스터의 전통적 연구과제중 하나인 시스템의 고성능화를 이룰 수 있게 된다.

· 본 연구는 한국과학재단 목적기초연구(R01-2003-000-10627-0) 지원으로 수행되었음
† 비 회 원 : 삼성전자(주) DVS사업부 시스템 프로그래머
syrinx@dcclab.sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터학과 교수
parksy@ccs.sogang.ac.kr
논문접수 : 2003년 3월 27일
심사완료 : 2003년 8월 5일

하지만 시스템의 고가용성을 달성하기 위한 소프트웨어를 효율적으로 개발하기 위해서는 유저 수준에서의 고가용성 보장을 위하여 사용되어온 그룹 통신 시스템(예: Horus[1,2], Ensemble[3], Lansis[4] 등)이 필요하다. 그러나 커널 수준의 클러스터 소프트웨어를 개발하기 위해 활용할 수 있는 커널 수준의 그룹 통신 시스템은 전무한 형편이고, 현재 개발된 그룹 통신 시스템은 사용자 수준에서 개발되었기 때문에 커널 수준에서 사용하기에는 무리가 있다.

본 논문에서는 기존 그룹 통신 패키지들이 가진 그룹 동작기능을 향상시켜 커널 수준에서 구현된 커널 수준 통신 시스템인 KCGCS(Kernel-level Cluster Group Communication System)를 설계하고 구현하고자 한다. 사용자 수준에서 개발된 기존 통신 시스템들을 커널 수준에서 그대로 구현할 수 있는 것은 사실이나, 현재까지 개발된 여러 그룹 통신 시스템은 신뢰성과 확장성에서 여러 문제점들을 드러내고 있다. 예를 들어 상술한 Ensemble 시스템은 특정 프로세스에 의존적이기 때문에 그룹 통신 시스템이 오히려 중앙(core) 프로세스의 장애 상황에 의해 통신이 단절되는 문제를 내포하고 있고, 핫빗 메커니즘에 확장성이 고려되지 않아 많은 개수의 노드를 사용하는 클러스터 시스템에 필요한 확장성을 보장해주지 못하는 경우도 있다. 또한 장애 상황 발견 시간의 변화가 커서 장애 상황 발견 시간에 대한 보장이 어렵다.

KCGCS는 기존 그룹 통신 패키지들의 기능 및 특성은 물론이고, 기존 그룹 통신 시스템들의 문제점이라 할 수 있는 신뢰성과 확장성을 확보하였다. 분산 코디네이터(coordinator)를 사용하여 고가용성과 신뢰도를 향상시켰으며, 2단계 멤버 정보 테이블을 사용해 신뢰성 확보를 위해 복제되는 멤버 정보량을 최소화 하였다. 또한 브로드캐스트 형태의 통신을 최소화하고 경량화한 프로토콜을 사용해 확장성을 고려하였다.

본 논문의 구성은 다음과 같다. 2장에서는 그룹 통신에 대한 기존의 연구결과 및 문제점들을 살펴본다. 3장에서는 KCGCS의 기능 및 특성들을 설명하며, 4장에서는 KCGCS의 성능을 측정하기 위하여 대표적인 그룹 통신 시스템인 Ensemble과 비교한다. 5장에서는 결론을 내리고 향후 연구 과제에 대해 논의한다.

2. 관련 연구

2.1 그룹 통신 시스템의 개요

KCGCS는 커널 수준에서 개발된 그룹 통신 시스템이다. 현재까지 개발된 여러 그룹 통신 시스템들은 모두 도메인을 사용자 수준으로 하고 있기 때문에 KCGCS와의 직접 비교를 할 수 있는 제품은 없는 실정이다. 그러나 그룹 통신 시스템 자체의 개념은 도메인 독립적이지 않고, 사용자 수준과 커널 수준 모두 비슷한 개념으로 사용되기 때문에 KCGCS가 지원하고 있는 커널 수준 그룹 통신 시스템의 개념 자체를 설명하기 위해서는 기존의 그룹 통신 시스템들이 가지고 있던 목표와 특징들을 논의하는 것으로 충분하다고 본다.

2.1.1 그룹 통신의 목표

그룹 통신 시스템은 분산 응용 프로그램들에서 프로세스 그룹 간의 신뢰성을 확보하고, 예기치 않은 다운에 효과적으로 대처하기위한 시스템을 개발하기 위해 만들어졌다.

그룹 통신 시스템은 대개 프로세스의 하위 레이어나 라이브러리 형태로 제공 되며, 그림 1의 (b)와 같이 구성 프로세스들에게 위치 투명성(location transparency)을 제공한다. 또한 각 노드 혹은 프로세스들을 그룹으로 관리하여 송수신 타입의 메시지 전송과 함께 그룹 내 브로드캐스트, 멀티캐스트 스타일의 통신도 지원한다. 또한 상태를 점검하여 동작하던 노드의 예기치 않은 다운이나 동작하지 않던 노드의 재 진입 등을 모니터링하여 임무 수행에 지장이 없도록 한다. 따라서 그룹 통신 시스템의 개발 목표는 분산 시스템의 신뢰성 있는 통신을 제공하고, 시스템의 고가용성을 보장하는 것이라고 할 수 있다.

이러한 그룹 통신 시스템의 기능은 크게 장애 상황 발견 기능과 멤버 관리 기능으로 나누어 볼 수 있다. 본 논문에서는 이 두 가지 기능에 초점을 두어 그룹 통신 시스템을 설명하며, KCGCS를 설계하고, 또한 이 두 가지 관점에서 성능을 분석할 것이다.

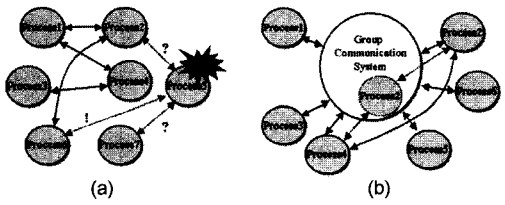

그림 1 추상화된 그룹 통신 시스템의 개념도

2.1.2 장애 상황 발견 기능

핫빗의 기본 개념은 핑퐁(ping-pong) 시스템과 비슷하다. 그림 2와 같이, 상태를 파악하기 위한 핫빗 메시지를 보내어 상태를 파악할 필요가 있는 대상에게 보내면, 그 대상은 자신이 정상으로 작동하고 있다는 메세지로 대답한다. 만약 이 메세지가 정해진 시간 안에 돌아오지 않는다면 핫빗 시스템은 상대방이 정상 상태가 아니라고 판단하게 된다.

대표적 장애 발견 메커니즘인 가쉽(Gossip)[5,6] 메커

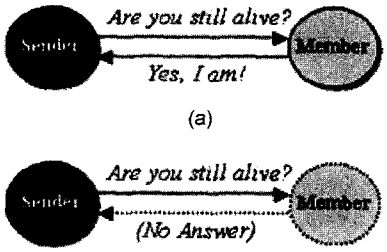

그림 2 핫빗 시스템

니즘은 Isis[7] 이후에 Transis[8]에서 도입하였으며 그 후 Horus와 Totem[9]에게 영향을 주게 되었다. Horus에 이어 개발된 코넬 대학교의 Ensemble 역시 이 가쉽 메커니즘을 장애 발견 메커니즘으로 사용하고 있다.

가쉽의 동작 원리는 다음과 같다. 모든 멤버는 핫빗 카운터(heartbeat counter)라고 불리우는 정수를 포함하여 알려진 멤버에 대한 멤버 테이블을 구성한다. 매 핫빗 시간마다 모든 멤버는 자신의 핫빗 카운트를 1씩 증가시키고 자신의 리스트에 있는 멤버들 중 무작위로 선택된 멤버들에게 핫빗 리스트를 보낸다. 즉 자신의 뷰(view)가 핫빗 메시지가 된다. 이 핫빗 메시지를 받은 멤버는 새로 받은 핫빗 리스트와 자신의 멤버 테이블을 비교하여 양쪽을 통합하고, 핫빗 리스트안에 속해 있던 멤버들의 핫빗 카운터를 각각 가장 큰 숫자로 바꾼다. 즉 핫빗 리스트를 받은 시점에서 자신의 뷰 안에 있는 모든 멤버가 살아 있다고 생각하는 것이다. 모든 멤버는 이러한 방법으로 자신을 제외한 모든 멤버들에 대해 마지막 핫빗 카운터가 증가된 시간을 테이블 안에 유지한다. 일정한 시간 안에 핫빗 카운터가 증가되지 않은 멤버가 있다면 그 멤버는 장애 상황에 빠진 것으로 간주한다.

2.1.3 멤버 관리 기능

멤버를 관리하기 위한 그룹 동작기능에는 새로운 그룹을 만드는 그룹 생성(create)(그림 3의 (a)), 생성된 그룹에 참가하여 기존 그룹 멤버들과 통신을 시작하는 그룹 참여(join)(그림 3의 (b)), 그룹에서 빠져나와 통신을 그만두는 그룹 탈퇴(leave)(그림 3의 (c)), 그룹을 없애는 그룹 파괴(destroy)(그림 3의 (d)) 등이 있다. 그룹을 구성하는 멤버는 개념에 따라 노드가 될 수도 있고, 프로세스가 될 수도 있으며, 쓰레드(thread)가 될 수도 있다. 이러한 멤버들의 정보를 유지하기 위하여 대부분의 그룹 통신 시스템은 코디네이터(예: 각 그룹의 코디네이터 역할을 하는 프로세스)를 두고 있으며 코디네이터가 중앙 집중식으로 구현되어 있어서 신뢰성에 문제가 있다.

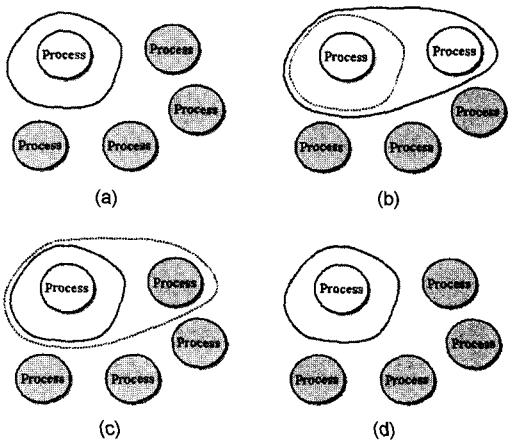

그림 3 그룹 동작기능

2.2 기존 그룹 통신 시스템의 문제점

그룹 통신 시스템의 기능은 크게 장애 상황 발견 기능과 멤버 관리 기능으로 나눌 수 있다. 장애 상황 발견 기능면에서 Ensemble을 비롯해 Isis 이후에 개발된 대부분의 그룹 통신 시스템은 가쉽 메커니즘을 장애 상황 발견 시스템으로 사용하고 있고, 이 메커니즘은 임의의 한 멤버에게 자신의 멤버 테이블을 전달하는 뷰 전달(flooding) 방식 때문에 노드 숫자가 늘어남에 따라 장애 상황 발견 시간을 보장하기 어려워지게 된다. 또한 뷰 내부에 멤버 리스트를 포함하기 때문에 그룹 구성원이 늘어남에 따라 핫빗 리스트의 크기 자체가 커지는 확장성(scalability)의 문제가 있다. 그 뿐 아니라 장애 상황 발견 시간의 변화가 커서 상황 발견 시간에 대한 보장 또한 어려워진다.

멤버 관리 면에서 볼 때, 기존 그룹 통신 시스템들은 중앙화 된 코디네이터인 가쉽에게 멤버 관리를 위한 통신을 의존하고 있기 때문에, 가쉽 자체의 동작 실패에 따른 장애 상황 문제를 내포하고 있으며 이는 멤버 관리의 신뢰성 문제로 연결된다.

3. 커널 수준 그룹 통신 시스템의 설계 및 구현

3.1 KCGCS의 개요

KCGCS는 커널 수준 응용 프로그램들, 특히 클러스터용 커널 수준 응용 프로그램들과 시스템 소프트웨어들의 고가용성을 지원하기 위해 제작된 커널 수준 그룹 통신 모듈이다. KCGCS는 기존 사용자 수준의 그룹 통신 시스템들의 기능들을 채용하면서, 나아가 기존 시스템들 보다 향상된 확장성과 신뢰성을 갖추도록 설계되었다. KCGCS를 이용하기 위해서는 공개된 KCGCS의 외부 응용 프로그램 인터페이스를 사용하여야 하며, 현재는 통신 모듈로서 커널 수준 통신 시스템인 KCCM

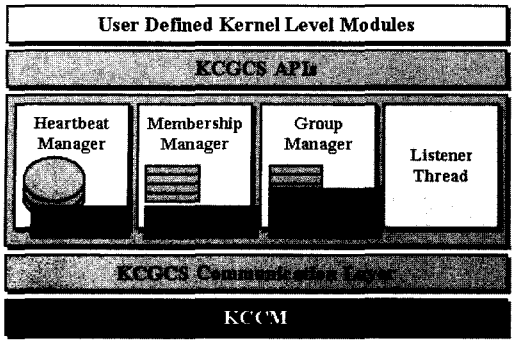

그림 4 KCGCS의 구조

(Kernel-level Cluster Communication Module)[10]을 사용하고 있다.

KCGCS는 초기화되면서 그림 4에서 보여주고 있는 것처럼 관리자 쓰레드라 불리우는 최초의 4개의 커널 쓰레드를 생성하며, 그것은 각각 핫빗 관리자(heartbeat manager), 멤버쉽 관리자(membership manager), 그룹 관리자(group manager), 리스너 쓰레드(listener thread)이다. 각 쓰레드들은 고유의 멤버 번호를 부여받고 메시지에 대해 수신 상태로 대기하며, 리스너 쓰레드는 하부의 통신 레이어로부터 도착한 메시지를 해당 쓰레드에게 전달한다. 모든 쓰레드는 각 메시지의 타입별로 수행하는 역할이 달리 정의되어 있다.

리스너 쓰레드는 각 노드에 도착하는 KCGCS 메시지를 받아들이고, 해당 메시지의 수신자 부분을 살펴본 후 행동을 결정한다. 멤버 관리자는 멤버 테이블이라는 자료구조를 유지하면서 월드 그룹이 아닌 일반 멤버 그룹의 정보를 관리한다. 그룹 관리자는 월드 그룹에 대한 정보를 유지하며, 각 멤버 그룹의 코디네이터에 대한 정보를 월드 그룹 멤버 테이블에 등록한다. 핫빗 관리자는 KCGCS가 시작 되었을때 핫빗 메시지를 브로드캐스트 하여 새로운 노드가 월드 그룹에 참여할 수 있도록 하고, 주기적으로 핫빗 메시지를 생성하여 장애 상황을 발견한다. 다음은 KCGCS의 구체적 특징들을 요약한 것이다.

3.1.1 커널 수준 그룹 통신 시스템

KCGCS의 기본 설계 목표는 커널 수준의 응용 프로그램들과 소프트웨어, 그리고 수정된 운영체제 등이 활용할 수 있는 커널 수준의 그룹 통신 시스템을 제공하는 것이다. 기본 목표에 따라 KCGCS는 리눅스 커널 버전 2.4에서 단일 커널 모듈로서 제작되었다.

커널 수준의 통신 시스템은 근본적으로 사용자 수준과 커널 수준의 경계 교차 오버헤드를 피하고, 직접적인 커널 자원 접근이 용이하기 때문에 고속화가 가능하다. 고속화된 통신이 가능하다는 것은 그 만큼 같은 핫빗 시간(heartbeat rate)을 가지고도 더 빠른 장애 상황 발견이 가능하며, 기존의 그룹 통신 시스템이 갖추고 있던 것 보다 고속의 그룹 동작 기능은 물론, 커널 수준의 사용자 프로그램에게 뷰 체인지 상황을 더 먼저 보고할 수 있다는 뜻이 된다. 본 논문에서 개발된 KCGCS는 사용자 수준에서 개발된 기존 그룹 통신 시스템보다 40~60% 정도 향상된 장애 상황 발견 및 그룹 동작 기능의 성능을 보였다(4장 참조).

3.1.2 향상된 신뢰성

그룹 통신 시스템의 개발 목적중 하나가 장애 상황을 감지하는 것뿐만이 아니라 그를 발견해내고 사용자에게 보고하는 것에 있기 때문에 신뢰도 높은 장애 상황 발견은 물론, 그 자신이 신뢰성을 갖추어 오히려 그룹 통신 시스템에 의해 응용 프로그램이나 소프트웨어가 불안정해 지는 것을 피해야 한다. KCGCS는 고속화된 그룹 통신 외에도 분산 코디네이터를 사용하여 더욱 신뢰성을 높일 수 있었다.

3.1.3 향상된 확장성

클러스터가 MPP에 비해 각광을 받고 있는 이유 중 하나가 바로 간단한 작업을 통하여 프로세서 혹은 노드를 추가하고, 이러한 방식을 통해 필요한 성능 향상의 효과를 쉽게 볼 수 있다는 점에 있다. 따라서 모든 클러스터 기반 소프트웨어는 이러한 확장성을 고려하여 설계되어야 한다.

KCGCS는 브로드캐스트 기반 혹은 플러딩(flooding) 방식을 사용하여 확장성에 문제점을 보여 왔던 기존의 그룹 통신 시스템과는 달리 링 구조의 핫빗 방식과 경량화 된 핫빗 메시지를 사용해 확장성을 보장하였으며, 또한 코디네이터의 동적인 이동(dynamic migration)을 위해 멤버 정보를 단순히 복제(replication)하지 않고 2중 구조의 멤버 정보 테이블을 관리함으로서 노드에 복제된 멤버 테이블의 크기를 최소화 하여 확장성을 향상시켰다.

3.1.4 이식성

통신 시스템에 있어서 이식성은 같은 환경에서 여러 하드웨어나 소프트웨어 레이어 혹은 모듈을 사용함에 있어서 별 다른 소스의 수정 없이 기존의 소프트웨어를 새로운 환경에 활용할 수 있는 측면과, 다른 하드웨어에 대한 적응성을 의미하는 저 수준 코드 수정의 측면을 고려할 수 있다.

KCGCS는 내부 구조에 통신 레이어를 따로 마련하여 KCGCS가 개발될 당시 사용하는 통신 시스템으로부터 쉽게 분리가 될 수 있으며, 따라서 KCGCS를 그룹 통신 모듈로서만 활용하기를 원하는 경우 간단한 수정만으로도 새로운 통신 모듈과의 결합을 이룰 수 있다. 또한 KCGCS는 커널 모듈로 제작되었기 때문에 직접적인

커널 코드 수정을 필요로 하지 않았기 때문에 소스 코드의 관리가 간편하며, 수정 및 배포도 용이하게 설계되었다.

3.2 장애 상황의 발견

3.2.1 핫빗 및 장애 상황 발견 메커니즘

그룹 통신 시스템에서 활용되었던 핫빗 메커니즘은 크게 나누어 세 가지가 있다. 일대일 통신을 이용한 브로드캐스트 기반 핫빗과 IP 멀티캐스트를 사용한 브로스캐스트 기반 핫빗, 그리고 가쉽에서 사용된 플러딩 방식의 핫빗이 있다.

일대일을 기본으로 한 브로드캐스트 기반 핫빗은 구현이 쉽다는 장점이 있으나, 각 멤버의 브로드캐스트를 통한 메시지 전달의 횟수가 많아져 확장성에 문제를 보인다. 멤버 하나가 핫빗을 하기 위해 필요한 패킷의 개수를 살펴보면, 자신을 제외한 모든 멤버에게 핫빗 메시지를 전달한다고 할 때, $(n-1)$개의 패킷이 필요하다. 이 핫빗 요구에 대한 응답이 필요하므로 한번의 핫빗이 완성되기 위해 $2(n-1)$개의 패킷이 요구한다. 결국 모든 멤버가 핫빗 메시지를 발송한다고 가정하면 $2n(n-1)$개의 패킷이 한번의 핫빗 시간 안에 필요한 셈이 된다.

IP 멀티캐스트를 사용한 브로드캐스트 기반 핫빗은 상술한 일대일을 기본으로 한 브로드캐스트 기반 핫빗과 마찬가지로 확장성에 문제를 보인다. 즉 핫빗 요구 메시지 자체는 IP 멀티캐스트를 이용하여 전달되지만 그 응답 메시지는 역시 각각 돌아오게 되므로 한번의 핫빗이 완성되기 위해 n^2개의 패킷이 필요하게 된다. 또한 IP 멀티캐스트 기반 핫빗은 클러스터를 구성하는 모든 노드 및 라우터가 IP 멀티캐스트를 지원해야 하므로, 로컬 네트워크로 묶여진 클러스터를 제외한 광역 네크워크(WAN: Wide Area Network)에서 활용하기에는 적합하지가 못하다.

가쉽 스타일의 플러딩 방식 핫빗은 핫빗의 요구와 요청이 따로 분리되지 않고 자신의 뷰를 핫빗 메시지로 대신함으로서 패킷 개수는 가장 적다는 장점이 있다. 즉 각 핫빗 타임에 자신의 뷰를 임의의 멤버 하나에게 발송하기 때문에 한번의 핫빗 타임 안에 필요한 패킷의 개수는 n개가 된다. 그러나 이러한 방식을 따를 때에는 노드가 확장됨에 따라 멤버에 장애 상황이 발생하였다고 판단되는 시간을 결정하기가 매우 어려워진다. 이것은 장애 상황 발견 시간의 변화 정도는 장애 상황 발견 시간에 대한 보장이 어렵다는 단점으로 연결된다.

그 뿐만이 아니라 가쉽 스타일의 핫빗 방식은 핫빗 메시시 내부에 자신의 뷰를 포함시키고 있다. 이는 핫빗 시간에 모든 노드가 핫빗 메시지를 한 번에 송신하기 위하여 필요한 네트워크 대역폭 $Bandwidth_{req}$를 생각해 보면, 멤버가 $Member_1$, $Member_2$, $Member_3$, … $Member_N$이라고 할때 각각의 핫빗 메시지 크기 $Size(Member_n)$와 메시지 개수 $Num(Member_n)$에 대하여 $Bandwidth_{req}$는,

$$Bandwidth_{req} = \sum_{n=1}^{N}(Size(Member_n) \times Num(Member_n)) \quad (1)$$

이다. 따라서 가쉽 스타일의 핫빗 방식은 그룹의 규모가 증가하면 함께 핫빗 메시지 자체의 크기가 증가하는 확장성의 문제를 가지고 있다(표 1 참조).

KCGCS는 형태는 같으나 두 가지 종류의 핫빗을 사용한다. 하나는 월드 그룹 멤버들 사이에서 동작하는 핫빗이며, 다른 하나는 그룹 단위로 동작하는 핫빗이다. Ensemble 시스템은 핫빗을 브로드캐스트하기 때문에 핫빗 자체의 제어 오버헤드가 큰 편이다. 그러나 KCGCS는 일대일 핫빗을 사용하며, 그를 위해 핫빗 그룹을 링 형태로 구성한다.

링 구조 핫빗은 표 1에 나타난 바와 같이, $O(n)$의 필요 네트워크 대역폭을 가지면서도 최소화된 $2n$개의 패킷 개수만을 요구한다. 기존 다른 그룹 통신 시스템보다 확장성면에서 개선되었다고 할 수 있을 것이다.

3.2.2 브로드캐스트 메시지의 복구

KCGCS는 IP 멀티캐스트 등을 사용하지 않고 일대일 통신을 이용한 브로드캐스트를 사용한다. 이 때, 브로드캐스트 중인 프로세스에 장애 상황이 발생하여 모든 노드가 브로드캐스트 된 메시지를 받지 못한 경우가 생길 수 있다. 이런 경우를 감안하여 KCGCS는 비안정화 메시지 큐(unstable message queue)라는 자료구조를 이용해 브로드캐스트 메시지를 모든 그룹 멤버들이 수신할 수 있도록 한다.

표 1 핫빗 방식에 따른 핫빗 패킷의 개수와 필요 네트워크 대역폭

구 분	패킷 개수	필요 네트워크 대역폭	시간 복잡도
일대일 통신을 이용한 브로드캐스트 기반 핫빗	$2n(n-1)$	$2hn(n-1)$	$O(n^2)$
IP 멀티캐스트를 이용한 브로드캐스트 기반 핫빗	n^2	hn^2	$O(n^2)$
가쉽 스타일 핫빗	n	$n(h+vn)$	$O(n^2)$
링 구조 핫빗	$2n$	$2nh$	$O(n)$

※ 단 n은 노드의 개수, h는 핫빗 메시지 헤더 크기, v는 뷰 레코드 혹은 엔드포인트의 크기

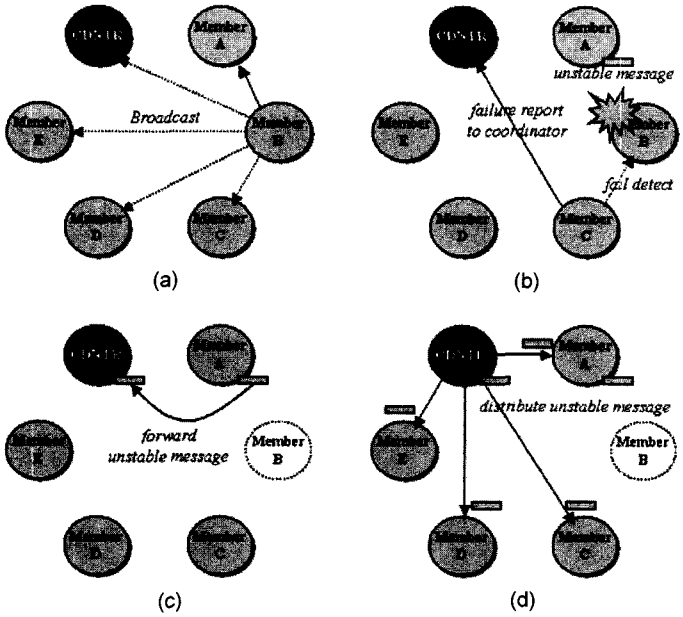

그림 5 비안정화 메시지의 전달과정

브로드캐스트 타입으로 메시지를 전달하기 위해서, KCGCS는 통신 레이어에서 브로드캐스트 타입 메시지를 전송한 후, 해당 메시지가 모든 그룹 멤버들에게 전송되었음을 알리는 안정화 커밋(commit) 메시지를 전송한다. 전송된 모든 메시지는 각 노드의 리스너 쓰레드가 관리하는 비안정화 메시지 큐에 저장된다. 이 비안정화 메시지 큐에 저장된 비안정화 메시지들은 안정화 커밋 메시지가 수신된 후 큐에서 삭제된다. 모든 멤버들은 서제스티드(suggested) 뷰 타입 메시지가 코디네이터로부터 도착하면, 자신의 비안정화 메시지 큐에 저장되어 있는 모든 메시지가 비안정화 상태에 놓여 있다고 판단하고, 이를 비안정화 타입 메시지로서 코디네이터에게 전달한 후 서제스티디 뷰를 받아들였다는 신호로서 플러쉬드(flushed) 타입 메시지를 코디네이터에게 전송한다. 코디네이터는 모든 멤버로부터 플러쉬드 타입 메시지가 도착한 후, 받아 두었던 비안정화 타입 메시지를 비안정화 메시지 복구 타입 메시지로 바꾸어 멤버들에게 전송한다.

예를 들면, 그림 5의 (a)의 경우, 멤버 B가 메시지를 브로드캐스트 하기 시작했으나, 멤버 A에게 메시지를 전송한 후 장애 상황에 빠졌다고 가정하자. 그림 5의 (b)와 마찬가지로 멤버 A는 브로드캐스트 타입 메시지를 받으면 그 메시지의 복사 본을 비안정화 메시지 큐에 저장한다. 멤버 B의 장애 상황 발생이 멤버 C에 의하여 발견되고, 멤버 C는 현재의 상태를 코디네이터에게 보고한다. 코디네이터는 뷰 체인지 메커니즘에 따라 새로운 뷰를 생성한 후 서제스티드 뷰를 멤버들에게 전송한다. 서제스티디 뷰를 받은 멤버 A가 그림 5의 (c)와 같이 자신의 비안정화 메시지 큐에 들어 있던 브로드캐스트 메시지를 코디네이터에게 전달하고, 코디네이터는 모든 멤버로부터 플러쉬드 메시지를 받은 후 멤버 A로부터 전달 받은 비안정화 메시지를 전체 멤버에게 전달한다(그림 5의 (d)).

이러한 메커니즘을 통해 KCGCS의 모든 멤버는 브로드캐스트를 하던 중에 장애가 발생한 노드로부터 발생되었던 메시지를 받아볼 수 있게 된다. 만약 브로드캐스트에 들어갔으나 하나의 메시지로 발생시키지 않고 장애상태에 빠졌다면, 그 멤버는 메시지를 전송하지 않은 것으로 간주된다.

3.3 멤버 관리

3.3.1 동적인 분산 코디네이터(distributed dynamic coordinator)

그룹 통신 시스템에서의 코디네이터란 그룹 멤버들의 상태 정보를 저장하고 멤버들 사이의 통신을 조율하며 관장하는 기능을 한다. 대부분의 기존 그룹 통신 시스템들은 가쉽이라는 중앙화된 코디네이터를 사용하여 모든 그룹 통신을 관장한다. 그러나 중앙화된 코디네이터는 장애상황을 피할 수 없고, 또한 그 자체가 병목 현상을 일으킬 수 있는 요인이 된다.

KCGCS는 이러한 문제를 해결하기 위해 그룹 통신에

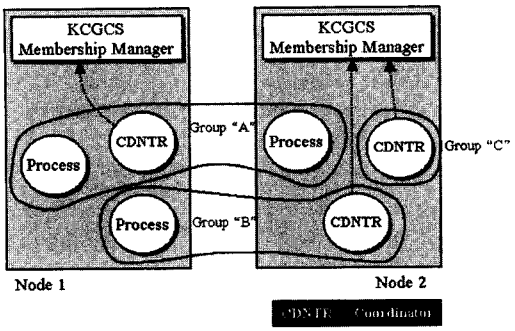

그림 6 분산 코디네이터와 멤버 관리자

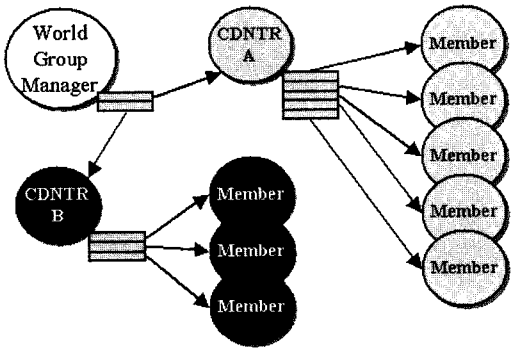

그림 7 월드 그룹과 2단계 멤버 정보

참여한 프로세스중 하나가 코디네이터로 설정되며, 각 멤버들은 개념적으로 그룹 내부의 멤버 정보를 복사해 가지게 된다. 실제로는 KCGCS 내부의 멤버쉽 관리자가 멤버 정보를 총괄하고, 해당 노드의 다른 프로세스들은 이 멤버 정보를 참조하게 되기 때문에 물리적으로 멤버 정보는 노드 하나에 하나씩 저장된다. 그림 6의 경우, 그룹B와 그룹C는 각기 다른 그룹의 코디네이터이나, 만약 멤버 중 코디네이터 프로세스가 다운되어 더 이상 코디네이터의 역할을 수행할 수 없게 될 경우, 나머지 노드들은 서로 통신을 하여 살아남아 있는 멤버들의 정보를 모으고, 상태가 활동적(active)인 멤버 중 새로운 코디네이터를 선정한다. 이러한 코디네이터의 동적인 이동(migration) 메커니즘을 통해 KCGCS 그룹의 멤버 프로세스들은 코디네이터의 다운 후에도 수행하던 일을 계속할 수 있는 고가용성이 보장된다.

3.3.2 월드 그룹(World Group)

KCGCS는 그룹에 참여할 가능성이 있는 모든 노드의 정보를 모아 월드 그룹이라는 전체 도메인을 구성한다. 초기화 된 월드 그룹은 단지 노드의 인터넷 주소 정도만을 정보로 가지고 있으나, 새로운 그룹이 생성되면 그 그룹의 코디네이터에 해당하는 노드의 정보를 수집하여 보관하며, 특정 그룹의 정보가 갱신될 때마다 월드 그룹 내의 모든 그룹 코디네이터들에게 메시지를 전송 해 서로의 정보를 갱신하게 한다. 그룹 멤버들의 정보가 그룹 내부에 복제되어 있는 것처럼, 월드 그룹의 정보는 각 그룹 코디네이터들에게 복제되어 있는데, 이렇게 2단계에 걸쳐 정보를 저장함으로 해서 노드 하나가 가지는 멤버 테이블의 크기는 최소화하면서도 효율적인 정보 복제가 가능하게 된다.

그림 7을 살펴보면, 그룹 A는 코디네이터를 포함하여 여섯 개, 그룹 B는 네 개의 멤버를 가지고 있다. 후에 코디네이터가 예기지 않은 장애상황에 처하게 되어 그룹의 다른 멤버가 코디네이터로 선출될 필요가 있을 경우를 위해 코디네이터가 가지고 있는 그룹 A의 멤버 정보는 코디네이터가 아닌 그룹 A의 멤버 다섯에게도 복제가 되어 저장된다. 코디네이터 B가 관장하는 그룹 B의 경우도 마찬가지로 그룹 정보는 멤버들 사이에 복제되어 저장되어 있다. 그러나 각 그룹의 코디네이터에 대한 정보는 월드 그룹 관리자(world group manager)가 관리하기 때문에 코디네이터 A를 비롯한 그룹 A의 멤버들은 코디네이터 B를 위시한 그룹 B의 멤버 정보를 관리할 필요가 없다. 그룹 B의 경우도 마찬가지이다.

만약 이러한 메커니즘을 사용하지 않는다면 멤버 그룹의 코디네이터가 속한 노드의 그룹 관리자가 그룹 A와 그룹 B의 정보를 관리해야 한다. 하지만 2단계에 걸쳐 멤버 정보를 관리하게 되면 관리해야 하는 그룹 멤버의 정보가 현격히 줄어 들 것이다.

4. 성능 평가

3장까지 커널 수준 그룹 통신 시스템인 KCGCS의 개발 동기와 설계 및 구현 부분을 살펴보았다. 그러나 1장에서 언급했던 바와 같이 현재 커널 수준에서 구현된 그룹 통신 시스템은 없기 때문에 4장에서는 가장 잘 알려져 있고 많이 쓰이는 그룹 통신 시스템인 Ensemble 시스템과의 비교를 통하여 KCGCS의 성능을 측정해 보고자 한다.

4.1 성능 평가 환경

본 연구에서는 100Mbps 랜으로 서로 연결된 13 노드 Pentium III 클러스터를 이용하여 성능을 측정하여 보았다. 앞서 언급한 바와 같이 KCGCS는 일대일 통신을 위하여 리눅스 기반 통신 모듈인 KCCM을 사용하였으며, Ensemble 시스템은 1.38 버전을 사용하였다. 공정한 성능 비교를 위해 13개의 모든 노드에는 각 1개씩의 멤버를 두었으며, 모든 멤버는 하나의 그룹에만 속하도록 설정하였다.

4.2 확장성 평가

그룹 통신 시스템은 클러스터의 고가용성 보장을 위

해 제작된 것이고, 클러스터의 장점 중 하나는 노드 추가나 삭제에 의한 쉬운 성능 증량에 있다. 따라서 그룹 통신 시스템은 클러스터의 구성 노드 개수가 증가하더라도 통신 지연 시간 증가는 최소화해야 한다. 만약 구성 노드 개수가 증가함에 따라 통신 지연 시간이 비약적으로 증가한다면, 사용자 프로그램이 필요로 하는 통신 성능을 제공해 줄 수 없게 되거나, 혹은 그로 인한 장애 상황 발견 시간(failure detection time)과 생존 상황 발견 시간(live detection time)의 증가로 인해 사용자 프로그램에 대해 최소 상태 변화 보고 시간을 보장해줄 수 없게 된다.

본 논문에서는 그룹 통신 시스템 확장성의 평가 기준으로 구성 노드 개수 증가에 따른 장애 상황 발견 시간과 생존 상황 발견 시간의 변화량을 선택하였다. 이 두 동작은 그룹 통신 시스템이 제공해야 하는 핵심 기능이며, 동작이 완료되기 위해서는 각 노드의 멤버들이 서로 통신을 하여 새로운 뷰에 대해 동의해야 하는 동작이기 때문에 구성 노드 개수 변화에 민감하다. 따라서 장애 상황 발견 시간과 생존 상황 발견 시간을 확장성의 평가 기준으로 선택한 것은 타당하다 할 수 있다.

4.2.1 장애 상황 발견 시간(failure detection time)

측정 방법

장애 상황은 실제로 장애 상황이 발견된 순간부터 그 장애 상황을 발견하여 모든 멤버들이 동의한 상황에서 사용자에게 뷰 체인지가 내려오는 시간까지로 하였다. 그러나 실제로 장해가 발생한 멤버는 뷰 체인지가 되는 순간 멤버 테이블에서 빠지기 때문에 뷰 체인지를 받을 수 없다. 따라서 장애 상황 발견 시간을 계산하기 위해서 코디네이터 노드에 시간 서버를 만들었다.

예를 들면, KCGCS의 경우 멤버가 코디네이터가 있는 노드의 시간 서버에게 시그널을 보낸 후, 즉시 동작을 끝낸다. 핫빗에 의하여 노드의 장애 상황이 발견될 것이며, 발견된 장애 상황이 코디네이터에게 보고 될 것이다. 코디네이터는 새로운 서제스티드 뷰를 내린 후 플러쉬드 메시지를 모아 뷰 체인지를 시작한다. 그리고 사용자 모듈로 뷰 체인지 메시지를 전송하는데, 이때 사용자 모듈은 코디네이터 노드의 시간 서버에 기록된 시간과 현재의 시간을 비교하여 장애 상황 발견 및 뷰 체인지에 소요된 시간을 계산한다. 실험에서 정의한 장애 상황 발견 시간에 대한 대략적 개념도를 그림 8에 나타내었다.

Ensemble도 이와 유사한 방법으로 만들어졌다. 시간 서버를 둔 후, Ensemble 라이브러리를 포함하여 동작 중이던 노드가 메시지를 시간 서버에 송신한 직후 동작을 끝낸다. 시간 서버가 있던 노드에서 뷰 체인지 메시지를 받은 후, 시간 서버가 기록한 시간을 비교하여 뷰

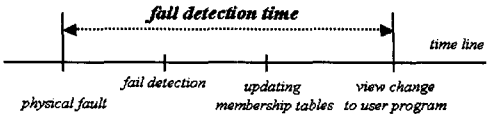

그림 8 장애 상황 발견 시간에 대한 대략적 개념도

체인지에 소요된 시간을 계산한다. KCCGCS와 Ensemble 시스템 양쪽 모두 핫빗 시간은 2초를 주고 성능 평가를 실시했다.

평가 결과

핫빗 시간을 2초라고 가정했을 때, 이론적으로 장애 상황 발견에 필요한 최소의 물리적 평균 시간은 1초이다. 이 물리적인 시간 때문에 평균 1초 이하로는 실험 결과가 나올 수 없었다.

실험 결과 KCGCS는 실험에 사용된 노드 숫자가 늘어남에 따라 Ensemble 시스템과 큰 차이를 보였다. 브로드캐스트 등의 통신 오버헤드가 거의 없는 경우, 즉 2개 노드 중 1개의 실패 상태와 같은 경우에는 1초의 물리적 필요 시간 때문에 별 큰 차이가 없는 성능을 보였으나, 노드의 숫자가 늘어남에 따라 양쪽의 차이가 점점 벌어지는 것을 확인할 수 있다. 이러한 이유는 Ensemble의 장애 상황 발견 시스템인 가쉽과 KCGCS의 장애 상황 발견 시스템의 동작 메커니즘의 차이에 있다고 보여 진다. 가쉽은 일정한 핫빗 시간 마다 멤버 테이블을 자신의 멤버 테이블 내에 있는 임의의 멤버에게 전송하고, 이 전송된 멤버 정보에 의거해 장애 발생 노드를 판단한다. 이때 전송되는 멤버의 정보량은 핫빗 메시지가 이미 뷰 정보를 포함하고 있기 때문에 멤버의 크기에 비례하여 증가한다. 반면 KCGCS는 링 모양 핫빗 그룹을 형성하고, 각 멤버가 하나의 멤버의 생존 상황을 보장하는 방식으로 설계되었다. 또한 일정한 크기의 핫빗 메시지를 사용하기 때문에 멤버의 숫자가 늘어나더라도 장애 상황 발견 시간에 큰 차이가 없다. 즉 KCGCS는 통신 오버헤드만이 증가하는데 비해, 가쉽은 발견 메커니즘이 필요로 하는 시간이 더욱 길어지기 때문에 그림 9의 결과가 나왔다고 할 수 있다.

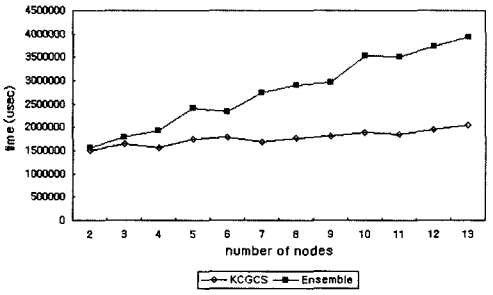

그림 9 장애 상황 발견 시간 비교

물론 통신 속도 면에서, KCGCS가 커널 수준에서 구현되었기 때문에 사용자 수준 오버헤드를 피할 수 있었다는 점은 고려해볼 수 있으나, 이렇게 노드 숫자에 크게 구애를 받지 않는 다는 점에서 Ensemble 시스템보다 확장성 면에서 더 뛰어나다고 볼 수 있을 것이다.

4.2.2 생존 상황 발견(live detection) 시간

측정 방법

새로운 멤버가 그룹에 추가되었을 때, 그 멤버는 초기에 자신이 알고 있는 멤버 혹은 노드에 브로드캐스트 메시지를 발송하여 자신의 생존을 알리게 된다. 이 점은 KCGCS와 Ensemble 시스템이 모두 같은데, 그 후 그 멤버의 생존을 확인한 그룹은 뷰 체인지를 내려 그 멤버를 그룹에 받아들인다.

Ensemble 시스템에는 월드 그룹이라는 개념이 없다. 따라서 성능을 평가하기 위해서는 1개의 임의그룹을 작성한 후, 멤버 리스트에 존재하는 멤버 프로세스를 실행시키는 것을 물리적 생존 시간으로 결정하였다. KCGCS는 월드 그룹을 생존 발견의 기준으로 활용하였다. 즉 멤버가 월드 그룹에 들어와 자신의 생존을 브로드캐스트 하는 시간을 생존 시간으로 결정하고, 그것을 위해 KCGCS를 사용하는 임의의 시험 프로그램 모듈을 제작하고 KCGCS의 그룹 관리자를 동작 상태로 만들어 놓았다. 앞서의 실험과 마찬가지로 각 노드에 1개씩의 멤버가 생성되었다.

두 번째 성능 평가 역시 4.2.1절에서 소개되었던 시간 서버를 활용하였다. 즉 특정 노드에서 시간 서버를 동작시킨 다음, 새로 깨어나는 멤버는 시간 서버에 신호를 보내 시간을 기록하게 한다. 그리고 시간 서버에서 동작하는 노드의 멤버가 뷰 체인지를 받고 그 시간을 기록하는 것으로 했다. 생존 상황 발견은 장애 상황 발견과 경우가 매우 다르다. 생존 상황 발견은 생존 멤버 스스로가 자신이 살아 있음을 알려진 멤버에게 브로드캐스트하는 것으로 시작하기 때문에 작동이 능동적이다. 즉 핫빗 시간에 의존적이지 않다.

평가 결과

그림 10의 결과에서 볼 수 있듯이, 생존 상황 발견 역시 KCGCS가 훨씬 좋은 성능을 보였다. 1초의 강제 시간이 필요하지 않기 때문에 커널 수준 통신과 유저 수준 통신의 성능 차이가 더욱 극명하게 드러났다.

새로운 멤버가 그룹에 들어오기 위하여 자신이 깨어났다는 내용의 메시지를 전체 노드에 브로드캐스트 한다는 점은 KCGCS와 가쉽이 같다. 그러나 역시 임의로 선택된 노드에게 멤버 개수에 비례하여 크기가 증가하는 핫빗 메시지를 전송하는 가쉽에 비해, 일정 길이의 경량화된 핫빗 메시지를 사용하는 KCGCS가 더 좋은 성능을 보인다고 보여진다.

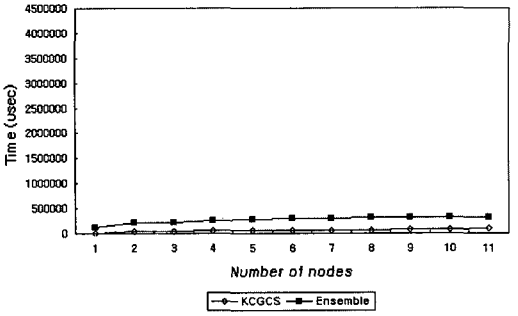

그림 10 생존 상황 발견 시간 비교

노드가 1개일 때 발견 시간이 아주 작은 것은, 실제로 노드 1개의 생존 발견이란, 새로운 그룹의 생성과 같은 뜻이기 때문이다. 즉 로컬에서만 뷰 체인지가 일어나며 외부로의 통신이 필요치 않기 때문에 필요 시간이 아주 작게 나타난다.

4.2.3 제어 메시지(control message)와 네트워크 대역폭(bandwidth)

측정 방법

확장성을 실험하기 위하여, 표 1에서 설명한 대로, 일대일 기반의 브로드캐스트를 이용한 핫빗과, IP 멀티캐스트 기반의 브로드캐스트를 이용한 핫빗, 가쉽, 그리고 KCGCS의 링 구조의 핫빗의 제어 메시지의 개수와, 제어 메시지를 전달하기 위해 필요한 네트워크 대역폭을 계산해 보기로 한다.

4.2.1절의 장애 상황 발견 시간 실험과 4.2.2절의 생존 상황 발견 시간에 대한 실험이 멤버 테이블이 변동되는 특별한 상황에 대한 실험이었다면, 이 실험은 장해나 생존 등 특별한 상황이 발생했을 때가 아니라 멤버 테이블에 변동이 없는 일반적인 경우를 고려하는 것이다. 장애 상황과 생존 상황은 언제든 발생할 가능성이 있는 것이나, 실상 아무런 문제가 발생하지 않은 정상 상태에서 클러스터가 동작할 확률이 더 높기 때문에 이러한 실험 또한 의미 있는 실험이 될 것이다.

그러나 장해나 생존 발견이 없다면 특별히 측정할 수 있는 기준 시간이 없다. 따라서 이번 절에서는 실험이 아니라 표 1에서 설명한 여러 핫빗 메커니즘에 따라 발생하는 제어 메시지의 개수와 필요 네트워크 대역폭을 계산하여 비교함으로서 네 가지 메커니즘에 대한 확장성을 검증할 것이다.

계산에 앞서, 표 1에서 정의한 핫빗 메시지의 헤더 크기와 엔드포인트의 크기를 각각 20바이트라고 가정한다. KCGCS의 메시지 헤더 크기는 18바이트 이며, 가쉽은 도메인 이름(domain name)이나 문자열로 저장된 IP 주소를 엔드포인트로 사용할 것을 권장하므로, 타당한 가정이라고 생각된다. 또한 필요 네트워크 대역폭은 3.2.1

절의 식 (1)을 사용하였다.

실험 결과

위의 가정에 따른 실험 결과가 그림 11과 그림 12에 나타나 있다. 제어 메시지의 개수를 계산한 그림 11의 경우, 일대일 기반과 IP 멀티캐스트 기반의 브로드캐스트를 이용한 핫빗에 비하여 가쉽과 링 구조의 핫빗의 제어 메시지 개수가 월등히 적은 것으로 나타났다. 그림 11의 결과를 보면, 매 핫빗 시간 때마다 핫빗 요구 메시지를 브로드캐스트하고 그 응답을 기다리는 브로드캐스트 기반 핫빗에 비해, 특정한 하나의 멤버에게 핫빗 메시지를 보내는 방법이 훨씬 경제적이라고 할 수 있다. 가쉽과 링 모양 핫빗을 비교하면, 가쉽은 뷰 메시지를 보내는 것에서 핫빗 과정이 끝나지만, 링 모양 핫빗은 응답을 받는다는 점에서 그래프에 나타난 바와 같이 가쉽이 유리하다고 볼 수 있다. 그러나 그림 12의 경우를 살펴보면, 가쉽이 IP 멀티캐스트 기반 브로드캐스트 핫빗과 비슷한 수준의 대역폭을 요구한다는 것을 알 수 있다. 이것은 가쉽의 경우 핫빗 메시지가 뷰를 포함하고 있고, 따라서 멤버가 증가함에 따라 핫빗 메시지의 크기가 커지기 때문이다. 이 경우에도 KCGCS의 링 구조 핫빗은 일정한 크기를 유지하여, 노드의 개수가 증가함에도 필요로 하는 네트워크 대역폭이 크게 늘지 않는 것을 그래프에서 확인할 수 있다.

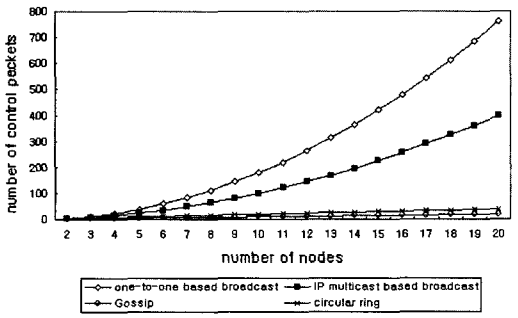

그림 11 제어 메시지 개수

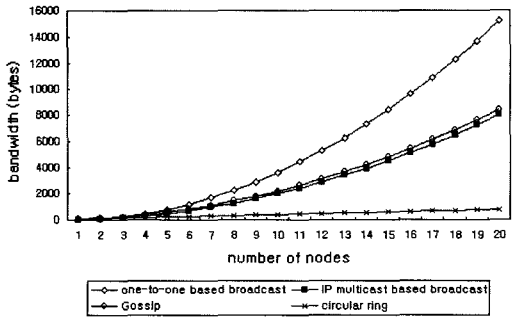

그림 12 제어 메시지의 요구 네트워크 대역폭

4.3 신뢰성 평가

Ensemble 시스템은 가쉽이라는 중앙 집중화된 프로세스에 의존적이다. 최초 환경 변수(environmental variables)에 등록된 엔드포인트의 한 곳에 가쉽 프로세스가 작동하면, 초기화과정에서 알려진 엔드포인트에 자신의 존재를 브로드캐스트하여 가쉽 프로세스와 접촉한다.

표 2 코디네이터의 개수와 단일점 장해상황의 가능성

	C = 1	C < N	C = N
KCGCS의 단일점 장해상황 가능성	X	X	X
Ensemble 시스템의 단일점 장해상황 가능성	O	O	X

그러나 두 개 이상의 가쉽 프로세스를 다른 엔드포인트에서 실행시킨다 하더라도, 중앙 집중성을 피할 수 있는 것은 아니다. 두 개 이상의 가쉽 프로세스가 실행되고 있더라도, 동작하는 가쉽 프로세스는 하나이며 나머지 하나는 동작하는 가쉽 프로세스가 장애 상황에 빠질 경우에 동작을 시작한다. 즉 전체 노드의 개수를 N이라 하고, 코디네이터(Ensemble 시스템 등의 경우 가쉽 프로세스)가 실행되어 있는 노드의 개수를 C라 할때, 각각의 C와 N의 관계에 따라 표 2와 같은 결과를 유추할 수 있다(단 C>0, N>0).

최상의 경우(best case), 즉 가능한 모든 엔드포인트에 가쉽 프로세스가 동작을 하고 있다면 이러한 문제가 발생하지 않는다. 그러나 가쉽이 동작하는 노드의 개수가 전체의 노드 개수보다 적다면, 노드에 장애 상황이 발생했을 시에 가쉽 프로세스가 Ensemble 시스템이 동작하는 노드중 어디에도 가쉽 프로세스가 동작하지 않는 경우가 발생할 가능성이 있다. 이러한 현상은 Ensemble 시스템이 장애 상황 발견을 위해 사용되는 가쉽 프로세스에 의존적이기 때문에 발생한다. KCGCS는 이러한 문제를 개선하기 위하여 모든 노드에 월드 그룹 관리자의 정보가 분산되어 있다. 신뢰성 확보를 위한 이러한 시도가 KCGCS의 확장성에 영향을 심각하게 주지는 않는다. KCGCS는 3.2.2절에서 설명한바와 같이 월드 그룹 관리자와 멤버 그룹 관리자가 각각 그룹 정보와 멤버 정보를 나누어 갖는 2단계 정보 테이블을 유지하여 멤버 정보의 복제를 최소화하였다.

5. 결 론

MPP와 SMP가 주류를 이루던 고속의 문제 해결 환경에 높은 성능에 낮은 단가를 가진 네트워크 장비를 갖춘 클러스터가 새로이 등장하면서 클러스터에 대한 관심이 증폭되고 있다. 클러스터에 대한 연구는 이미 많

이 진행되어, 지금은 고성능뿐만이 아니라, 고가용성까지 갖춘 클러스터 시스템을 개발하기 위한 연구가 진행중이다.

고가용성을 위한 그룹 통신 시스템에 대한 연구는 1980년대 중반부터 진행되었다. 지금까지 여러 그룹 통신 시스템들이 여러 과학자들에 의해 개발되어왔지만, 높아져가는 커널 수준 응용 프로그램과 소프트웨어에 비해 커널 수준에서 개발되어 커널 수준 프로그램들을 지원할 수 있는 커널 수준 그룹 통신 시스템들은 거의 없었다.

본 논문에서는 커널 수준의 그룹 통신 시스템이 갖추어야 할 요건들을 정의하고 리눅스 커널에서 동작하는 커널 수준의 그룹 통신 시스템인 KCGCS를 설계 및 구현하여 보았다. 경량화된 프로토콜을 사용하여 향상된 통신 속도로 인해 그룹 동작 기능이 고속화되었고, 분산된 유동적 코디네이터와 2중 구조 멤버 테이블 등의 아이디어를 이용해 신뢰도와 신뢰성을 향상시킬 수 있었다. 그 외에도 쉬운 인터페이스와 레이어적 접근을 통해 이식성과 조작성을 고려하였다.

KCGCS는 현재 기본적 그룹 통신 시스템이 갖추어야 하는 기능을 갖추고 있다. 그러나 KCGCS에는 보안에 대한 사항이 고려되어 있지 않으며, 현재에는 멤버의 단위를 프로세스로 제한하고 있기 때문에 쓰레드를 멤버로 활용하기 위해서는 다른 메커니즘의 도입이 필요하다. 또한 커널 기반의 소프트웨어가 가지고 있는 근본적인 이식성 문제라던가, 커널의 변경에 따른 소프트웨어의 변경 등을 최소화할 수 있는 방안도 고려되어야 할 것이다.

참 고 문 헌

[1] K. Birman, B. Constable, M. Hayden, J. Hickey, C. Kreitz, R. Renesse, O. Rodeh, W. Vegels, *"The Horus and Ensemble Projects: Accomplishment and Limitations,"* 2000.

[2] R. Renesse, K. Birman, S. Maffeis, *"Horus: A Flexible Group Communication System,"* 1996.

[3] *"Ensemble Tutorial,"* http://www.cs.cornell.edu/Info/Projects/Ensemble/doc/tut/

[4] O. Rodeh, *"The Design and Implementation of Lansis/E,"* 1997.

[5] S. Ranganathan, A. D. George, R. W. Todd, M. C. Chidester, *"Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters,"* 2001.

[6] R. Renesse, Y. Minsky, M. Hayden, *"Gossip-Style Failure Detection Service,"* 1998.

[7] K. Berman and R. V. Renesse, *Reliable Distributed Computing with the Isis Toolkit,* IEEE Computer Society Press, 1994.

[8] Y. Amir, D. Dolev, S. Kramer, and D. Malki, *"Transis: A Communication Sub-System for High Availability,"* FTCS Conference, July 1992.

[9] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, C. A. Lingley-Papadopoulos, *"Totem: A Fault-Tolerant Multicast Group Communication system,"* Communications of the ACM No4. Vol39, pp.54~63, 1996.

[10] 박동식, 이상균, 박성용, 권오영, 박형우, *"리눅스 클러스터를 위한 효율적인 커널 통신 시스템의 설계 및 구현"*, 한국 정보과학회 컴퓨터 시스템 연구회 추계 학술 발표회 논문집, pp.41~47, 2002.

이 상 균

2001년 2월 서강대학교 컴퓨터학과 졸업 2003년 2월 서강대학교 대학원 컴퓨터학과 졸업(석사). 2001년 2월~2002년 10월 매크로임팩트(주)에서 클러스터 소프트웨어 개발. 2003년 3월~현재 삼성전자(주) DVS사업부에서 시스템 프로그래머로 근무중. 관심분야는 클러스터 통신, 그룹 통신 시스템, High Availability, 리눅스 커널

박 성 용

1987년 서강대학교 전자계산학과 졸업 1994년 Syracuse 대학 Computer Science 석사. 1998년 Syracuse 대학 Computer Science 박사. 1998년~1999년 Bellcore 연구소 연구원. 1999년~현재 서강대학교 컴퓨터학과 부교수. 관심분야는 분산시스템, 클러스터 컴퓨팅 및 시스템, P2P 및 Grid 컴퓨팅