가상화 기법을 사용한 경계 없는 캐쥬얼 게임 서버 설계 및 구현†

(Design and Implementation of Borderless Casual Game Server using Virtualization)

김 성 백*, 이 재 동**
(SungBaek Kim and Jae-Dong Lee)

요 약 본 논문에서는 대규모 유저들이 같은 게임서버 채널에서 함께 게임을 못하는 문제를 해결하기 위해서 통합 가상화의 방식으로 링크 서버를 사용해 내부의 서버를 통합화하여 구축하는 방법을 제안한다. 온라인게임이 발전하면서 게임유저들은 친구, 파티들과 커뮤니티를 형성하게 되었고, 이들과 같은 온라인게임 공간에서 함께 게임을 하는 것에 가장 많은 재미를 얻는다. 하지만 물리적 서버의 한계로 인하여 동 시간에 같은 채널에서 게임을 할 수 있는 유저들의 숫자는 평균 100명 이하이며, 채팅, 쪽지, 친구 등과 같은 게임 커뮤니티로 연결이 가능한 유저수는 3,000 ~ 10,000명을 넘지 못하게 되어 있다. 본 논문에서 제안하는 방법으로 시스템을 구축함으로써 게임 유저들에게 보여주는 서버는 하나로 보여주고, 유저들은 게임에 접속한 모든 유저의 정보를 공유할 수 있다. 본 논문에서는 이렇게 구축된 서버를 테스트 클라이언트를 사용하여 온라인 서버의 품질 요구 조건인 확장성, 일관성, 최소지연성을 만족하는 것을 확인하였다.

핵심주제어 : 온라인 게임, 가상화, 통합 서버, 캐주얼 게임 서버

Abstract This paper suggests the way to build the integration of internal server by using link server with virtualization method. With the development of on-line games, game communities began to be formed by users, players get fun from playing games, sharing online space with friends, clans, and parties. However, the limit of physical server restrict the allows under average a hundred users to play at same time and place, and three thousand to ten thousand people to access to game community with chat, message, friends functions. By following the explanation of the method from this paper, this limit can be overcome. It will give the function to share information of all connected users in one displayed server. This paper demonstrates the key quality requirements of the server built by this way such as scalable architecture, consistency, and latency is fulfilled.

Key Words : Online Game, Virtualization, United Server, Casual Game Server

1. 서 론

온라인게임이 발전하면서 게임유저들은 친구, 클랜(파티)들과 커뮤니티를 형성하게 되었고, 이들과 같은 온라인(게임) 공간에서 함께 게임을 하는 것에 가장 많은 재미를 얻는다. 하지만 물리적 서버의 한계로 인

† 본 연구는 2009학년도 단국대학교 대학 연구비 지원으로 연구되었음.
* 단국대학교 컴퓨터학과
** 단국대학교 소프트웨어학과, 교신저자

해 동시간에 같은 공간(존)에서 게임을 할 수 있는 유저들의 숫자는 평균 100명 이하이며, 게임 커뮤니티(채팅, 쪽지, 친구)로 연결이 가능한 유저의 수는 3,000~10,000명을 넘지 못한다. 이에 동시 접속자가 많은 게임에서의 게임유저들은 자신이 원하는 다른 유저(친구, 클랜)들과 게임을 하기 위해서 같은 공간 내에서 게임이 가능한 서버를 찾아서 이동해야 한다.[1][2] 이러한 서버이동은 유저측면에서는 접속한 서버에서 다른 서버로 이동을 해야 하는 불편함이 생기고, 서버측면에서는 유저들의 서버 이동을 하기 위해서 불필요한 네트워크 부하가 생기게 된다. 서비스 측면에서는 이렇게 이동하는 유저들 수는 예측이 불가능하므로 서비스 안정성에 불안정한 요소가 된다.[3][4]

이에 본 논문에서는 대규모 유저들이 같은 게임서버 지역(존)에서 함께 게임을 못하는 문제를 해결하기 위해서 가상서버 방식으로 링크 서버를 사용해 내부의 서버를 통합화하는 환경을 구축한다. 이렇게 구축된 서버 시스템을 통해 게임 유저들은 하나의 서버에서 게임을 하는 것과 같은 느낌을 받게 되고, 유저들은 게임에 접속한 모든 유저의 정보를 공유하게 된다. 실제로는 수십 대의 서버들이 복잡한 분산처리를 통해 정보를 주고받지만, 유저에게 보이는 서버 그룹은 단 한 개를 유지하고 있는 것과 동일한 효과를 나타낼 수 있다. 곧, 유저들은 서버군을 선택하는 과정을 거치지 않고 게임을 진행할 수 있으며, 서버 그룹 안에 있는 각 서버들을 효율성 있게 구축해서, 게임 유저에게 질 좋은 서비스를 제공하는 서버를 구축할 수 있다. 본 논문에서 설계하는 경계 없는 게임서버를 “통합 서버(United Server)”라고 명명한다.

본 논문의 구성은 5개의 장으로 구성된다. 1장에서는 연구의 배경과 연구의 목적을 기술한다. 2장에서는 가상화, 게임서버의 구조 및 기술에 대해서 알아본다. 3장에서는 실제 통합서버를 설계하고, 4장에서는 서버의 구현 및 테스트 환경을 구축하고 통합 서버의 성능이 온라인 게임서버 품질 요구조건을 만족하는지 평가한다. 그리고 마지막으로 5장에서는 연구를 통한 결과를 분석하고, 향후 연구 계획에 대해 기술한다.

2. 관련 연구

2.1 가상화

가상화는 실제 존재하는 물리적 자원들을 논리적 자원들의 형태로 표시해 줌으로써 물리적 자원을 이용하는 사용자(구체적으로 애플리케이션)에게는 논리적 형태로만 나타내 주는 기술이다. 이러한 가상화에는 애플리케이션 가상화, 데스크톱 가상화, 서버 가상화, 스토리지 가상화, 네트워크 가상화가 있다.[5][6][7][12]

2.2 온라인 게임서버

네트워크 게임의 발전과 더불어 온라인 게임이 큰 인기를 끌고 있는데 이것은 온라인 게임이 다수의 클라이언트가 서버에 접속하여 서로간의 의사소통을 통해 게임을 즐길 수 있는 환경을 제공하기 때문이다. 온라인 게임은 유저들에게 원활한 플레이를 제공하기 위해서 게임서버 기술은 비약적인 기술발전을 해왔다.[3][8][9] 게임서버의 종류에는 단일서버, 멀티서버, 대칭서버, 비대칭서버가 있다.[5][8] 단일서버는 하나의 서버가 여러 개의 클라이언트를 수용하는 중앙 집중형의 대표적인 방식으로, 다수의 클라이언트를 처리하기에는 부담이 많으므로 여러 가지 편법을 적용하여 이를 미봉책으로나마 해결하고 있으며, 많은 수의 클라이언트를 감당하기 힘든 문제점이 있다. 또한 동시 접속자가 많아지면 서버 확장이 불가능하다. 하지만 이러한 구조는 구현이 쉬우며, 하나의 서버가 서버기능을 모두 수행하므로 서버간의 시간적 동기화문제나 교착상태, 동시성, 무결성등의 문제가 쉽게 해결될 수 있다.[10][11] 멀티서버 구조는 서버들을 분담 별로 나누어 처리하는 형태로서 로그인서버, 업데이트서버, 게임서버, 자동갱신서버, 트랜젝션 서버, 데이터베이스서버, 웹서버, 모니터 서버, 빌링 서버 등으로 구분된다. 서버들이 세부적으로 분할되어 처리 될수록 서버간의 전송 트래픽은 높아지지만 그 만큼 많은 수의 클라이언트를 수용할 수 있다. 멀티서버 구조는 수천명 이상의 접속자를 관리할 수 있으며, 접속자가 늘어날수록 서버를 확대함으로써 뛰어난 효과를 볼 수 있다.[4] 하지만 다수의 클라이언트를 수용하기 위해서는 대역폭이 큰 선로가 필요하다. 또한 서버간의 연결을 위해 내부 네트워크가 구성되어 있어야 한다. 멀티 서버는 분산 시스템 환경으로 인해서 발생할 수 있는 문제점들이 많으므로 많은 기술이 적용되어야 한다. 멀티 서버 구조는 서버간의 의존성에 따라 대칭 서버 구조

(Replicated Server)와 비대칭 서버(Non-Replicated Server)로 나뉜다. 대칭 서버 구조는 서로 다른 서버의 플레이어들 간의 의존성이 없을 때 사용하는 서버이다. 보드게임이나 턴 방식의 온라인 게임에 사용된다. 대칭 서버 구조는 최대 수용할 수 있는 클라이언트 수 만큼만을 할당 받아 관리하므로 서로 다른 서버에 접속한 플레이어들은 정보를 주고받기가 힘들다.[9] 비대칭 서버 구조는 서로 다른 서버의 플레이어들간의 의존성이 있을 때 사용하는 서버이다. 대부분의 머그 게임이 이 방식을 사용하고 있으며, 클라이언트의 존(Zone)의 위치에 따라 서버가 담당하여 처리하는 방식이 가장 많이 사용되고 있다. 하지만 같은 존에 많은 플레이어가 몰리면 그 존을 담당하는 서버만이 많은 트래픽을 발생시킨다는 문제점이 있다. 서버의 오버로드 분산을 위해 많은 방법들이 연구되고 있다.[9]

온라인 게임의 품질 결정 인자의 하나는 네트워크 성능이며, 대역폭과 지연성의 두 가지 척도가 있다. 대역폭은 게임의 확장성(scalability, 플레이어 수의 증가성)을, 지연성은 게임의 응답성을 좌우한다. 이들 양자는 하부 통신 링크가 제공하는 한정된 자원이며, 게임 정보를 한정된 자원을 활용하여 플레이어들에게 잘 분산시킴으로써 게임의 품질을 향상시키고자 함이 온라인 게임, 특히 게임 서버 기술이라 하겠다.[2] 최적의 네트워크 성능을 유지하기 위해서는 해당게임에 적합한 게임 통신 구조를 택하여야 하며, 고품질의 게임진행을 제공하기 위해서 게임서버는 다음과 같은 요구조건을 만족시켜 주어야 한다.[11]

2.3 게임서버 개발 프로그램 기법

대용량 데이터를 처리하기 위한 방법으로 해쉬 알고리즘과 동기화 기법이 사용된다.[13] 해쉬알고리즘은 데이터 무결성 및 메시지 인증 등에서 사용할 수 있는 함수로써 정보보호의 여러 메커니즘에서 이용되는 핵심 요소기술이다. 해쉬알고리즘이란 임의의 길이의 비트 열을 고정된 길이의 출력값인 해쉬코드로 압축시키는 함수이며, 암호학적 응용에 사용되는 대부분의 해쉬함수는 강한 충돌저항성을 지닐 것이 요구된다. 암호학적 해쉬알고리즘의 충돌 저항성은 디지털 서명에서 송신자외의 제 3자에 의한 문서위조를 방지하는 부인방지 서비스를 제공하기 위한 필수적인 요구조건이 된다. 이진 검색 알고리즘(binary search algorithm)은 정렬된 리스트에서 특정한 값을 찾는 알고리즘이다. 처음 중간의 값을 임의의 값으로 선택하여, 그 값을 찾고자 하는 값과 크고 작음을 비교하는 방식을 채택하고 있다. 처음 선택한 중앙값이 만약 찾는 값보다 크면 그 값은 새로운 최고값이 되며, 작으면 그 값은 새로운 최하값이 된다. 검색이 반복될 때마다 목표값을 찾을 확률은 두 배가 된다. 이진 검색은 분할 정복 알고리즘의 한 예이다.

게임 서버 개발을 위해서는 또한 동기화 기법이 필요하다. 동기화란 여러 쓰레드에서 공유 데이타를 접근하여 사용하고 있을 때 한 개의 쓰레드에서 공유 데이타를 사용하는 중에 쓰레드의 교체가 일어나서 공유 데이타가 온전하게 관리 되지 못하는 문제가 발생할 수 있다. 이 경우 문제가 발생하지 않도록 조정하는 기법을 동기화 처리라고 한다.[14]

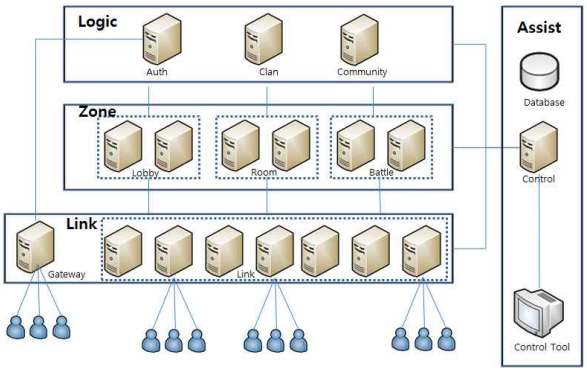
3. 통합 서버 시스템 설계

본 장에서는 클라이언트의 서버이동이 없는 통합 서버의 시스템을 설계 한다. 먼저 통합서버의 서버군을 정의/설계하고, 각 서버군의 시스템을 설계한다. 또한 이렇게 설계된 시스템을 온라인 게임 서비스품질 기준에 만족하도록 프로그램 속도 향상 기법을 제안한다.

3.1 기본 구성 요소

통합 서버를 설계하기 위해서 처리하는 일(JOB)로 서버군을 4가지 분류할 수 있다. 첫 번째로 연결서버군으로 클라이언트(게임유저: 이하 클라이언트로 통일)와 통신하는 외부 네트워크에 배치된 서버로써 게이트웨이서버와 링크서버가 있다. 두 번째로는 지역서버군으로 로비, 룸, 배틀 서버가 있다. 세 번째로는 로직서버군으로 인증, 클랜, 커뮤니티 서버가 있다. 마지막 네 번째로는 어시스트서버군으로 데이터베이스서버, 그리고 모든 서버의 시동, 멈춤, 상태저장을 하는 컨트롤서버가 있다. 컨트롤서버는 모든 서버의 동작(시작, 종료, 잠시멈춤, etc)을 컨트롤하는 서버로 이러한 컨트롤은 관제툴을 통해서 동작한다. 외부 네트워크로 연결되어 있는 서버(링크서버군)들은 공용 아이

피(Public IP)를 가지고 클라이언트와 직접적인 네트워크 통신을 한다. 그리고 내부 네트워크로 연결되어 있는 서버(링크서버를 제외한 모든 서버)들은 사설 아이피(Private IP)를 가지고 네트워크 통신을 한다.



<그림 1> 통합 서버 구성도

아래 <표 1>은 각각의 서버들의 기능표이다.

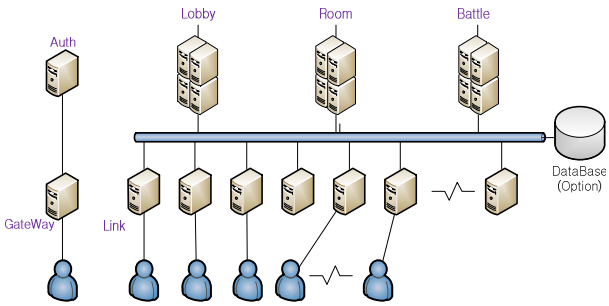
<표 1> 통합 서버 기능표

서버군	서버 이름	기능 설명
연결 (Connector)	게이트웨이 (Gateway)	• 클라이언트의 인증처리 후 부하 분산(로드밸런싱)
	링크 (Link)	• 클라이언트와 TCP 연결 • 클라이언트의 기본적인 정보 매니징
지역 (Zone)	로비 (Lobby)	• 마을, 상점 로직 처리 • 룸 리스트 정보 처리
	룸 (Room)	• 클라이언트 매치메이킹 • 룸 리스트 관리
	배틀 (Battle)	• 배틀처리 • 게임특성 - 게임 연산 • 게임특성 - 그래픽 연산
로직 (Logic)	인증 (Auth)	• 클라이언트의 정보 관리 • 상점 구입 관리 • 로그인 처리(퍼블리셔 연동단) • 빌링 처리(퍼블리셔 연동단)
	클랜 (Clan)	• 리스트 관리 • 멤버 리스트 • 매치메이킹
	커뮤니티 (Community)	• 채팅, • 귓속말, 쪽지 • 친구 시스템
어시스트 (Assist)	컨트롤 (Control) &관제툴	• 서버 상태 저장 • 서버구동(시작, 멈춤)
	데이터베이스 (Database)	• User Info (아이디, 닉네임, etc) • User Inventory, Shop • User Option • Community(Note, Friend) • Clan

3.2 시스템 설계

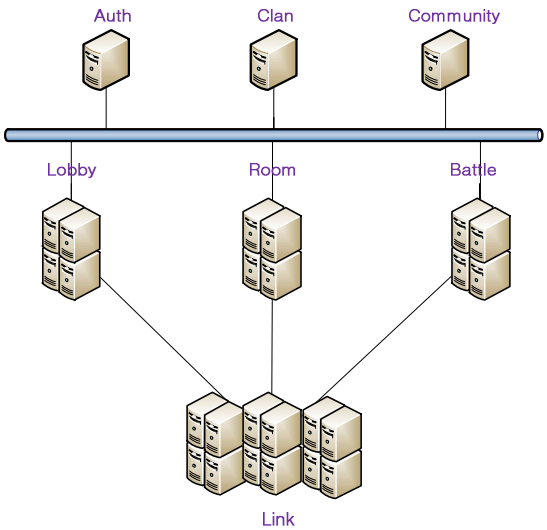
3.2.1 서버군 설계

연결(connector)서버군은 클라이언트와 연결을 가지고 있으며 대부분 동작은 클라이언트의 패킷을 지역서버군으로 전달하고, 지역서버군의 패킷을 클라이언트에게 전달하는 역할을 한다. 이 서버군에는 링크서버, 게이트웨이 서버가 있다.



<그림 2> 연결서버군 연결도

링크 서버(Linked Server)는 가장 앞단에서 클라이언트와 연결을 가지고 통신하는 서버이다. 내부 기능은 게임 서버군에서 오는 패킷을 클라이언트에게 전달하는 기능, 클라이언트와의 연결 유지 기능(TCP 하트비트 처리), 클라이언트의 옵션(Option) Load, Save 기능(Database와 연동)이 있다. 네트워크 연결은 외부 네트워크를 통해서는 클라이언트와 연결이 되어 있고, 내부 네트워크를 통해서는 지역서버군(Lobby, Room,

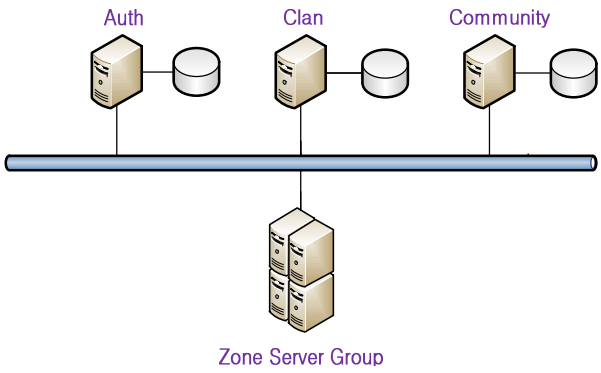


<그림 3> 기역 서버군의 연결도

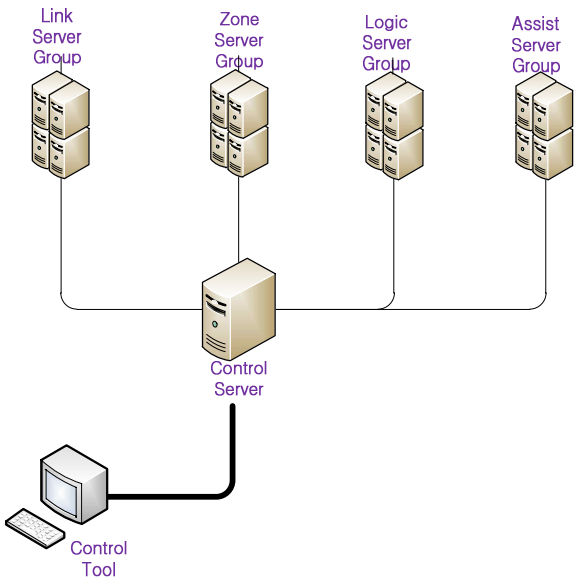
Battle)과 연결되어 있다. 게이트웨이 서버(Gate- way Server)는 클라이언트가 가정 처음 접속하는 서버이다. 먼저 클라이언트의 접속 요청을 받고 인증절차 후 클라이언트가 접속해야하는 링크서버의 정보를 클라이언트에게 전달한다. 게이트웨이 서버는 같은 연결서버군의 링크서버와 다르게 로직서버군의 인증서버와 연결되어 있으면서 클라이언트의 인증 요청을 처리한다.

지역(Zone) 서버군은 클라이언트의 대부분의 행동패턴을 처리한다. 이 서버군에는 세 가지 종류의 서버가 있는데 로비, 룸, 배틀 서버이다. 게임에서 클라이트들은 이 세 가지 지역에서만 존재할 수 있으며, 클라이언트가 지역을 이동할 때 마다 로직서버군과 통신하여 클라이언트를 각 서버군으로 이동시키는 역할을 하게 된다. 로비서버(Lobby Server)는 로그인을 한 유저들이 방에 들어가기 전 까지 있는 서버이다. 이 서버에서는 유저들은 클랜, 커뮤니티 기능을 사용하거나, 게임룸 리스트를 검색해서 자신이 원하는 게임룸을 찾아서 들어갈 수 있다. 그리고 채팅이나, 상점, 퀘스트 같은 MMORPG 마을에서 처리하는 모든 일을 담당한다. 룸 서버(Room Server)는 배틀에 들어가기 전까지의 기능, 즉 각 방에서의 채팅 및 방 속성 변경(인원수 제한, 게임 모드 변경, 강퇴, 방장 설정)을 담당하고 각 방의 리스트를 트랜스서버로 스트리밍을 한다. 또한 룸 서버에서는 로비서버에서 사용했던 기능 중 클랜, 커뮤니티, 상점기능을 사용할 수 있다. 배틀 서버(Battle Server)는 룸 서버에 있던 유저들이 배틀 준비를 마치고 실제 게임배틀을 시작한 상태이다. 배틀 서버에서도 게임리스트를 트랜스서버로 스트리밍을 한다. 게임 특성에 따라 난입을 불가능한 게임이라면 스트리밍을 할 필요는 없다. 각 룸에서는 전투 진행 패킷을 모든 유저들에게 맞게 전송하고 각 모드의 배틀을 컨트롤한다.

로직(Logic) 서버군은 클라이언트의 인증, 인벤토리, 상점, 커뮤니티시스템을 담당하는 서버로 구성되어 있다. 즉, 클라이언트의 지역이동, 배틀을 제외한 게임 전반적인 게임 시스템을 처리한다. 각 서버는 내부 서버 가장 최상단에 한 대씩 있으면서 각 서버들에게 정보를 전달해주는 역할을 한다. 인증서버(Auth Server)는 클라이언트의 인증(로그인, 로그아웃), 지역, 인벤토리를 담당하는 서버이다. 인증 서버는 게임서버와 내부 네트워크로 연결되어서 게이트웨이서버의 인증요청과 존서버군의 상점, 구입, 퀘스트, 존 이동, 인벤토리요청을 처리하는 역할을 한다. 결과 패킷을 존서버군, 연결서버군을 통해서 클라이언트에 전송한다. 또한 인증처리를 하면서 클라이언트들이 연결해야 할 링크서버의 인덱스를 발급하는 역할을 한다. 발급한 인덱스로 클라이언트들은 배당된 링크서버로 연결되면서 자연스럽게 부하 분산이 된다. 클랜서버(Clan Server)는 클랜을 담당하는 서버로서 클랜 리스트 관리, 각 클랜원 관리, 각 클랜 전적 데이터 관리, 클랜 매치 매이킹 등의 일을 한다. 또한, 지역(Zone) 서버군과 내부 네트워크로 연결되어 있으며, 클라이언트의 클랜 관련 요청을 처리한다. 커뮤티티 서버(Community Server)는 커뮤니티를 담당하는 서버로서 귓속말, 쪽지 관리, 친구 관리, 채팅 등을 담당한다.



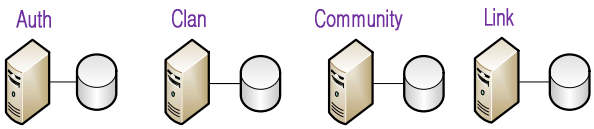
<그림 4> 로직서버군 연결도



<그림 5> 컨트롤 서버 연결도

또한, 클랜서버와 동일하게 지역(Zone)서버군과 내부 네트웍으로 연결되어 있으며, 클라이언트의 요청을 처리하고 결과 패킷을 지역(Zone)서버군, 링크서버를 통해서 클라이언트에게 전송한다.

어시스트(Assist) 서버군은 직접적으로 유저들의 요청을 처리하는 서버가 아닌 내부 서버군의 일을 도와주는 서버들의 집단이다. 컨트롤 서버(Control Server)는 모든 서버와 연결이 되어 있다. 각 서버들은 시스템 정보(CPU, Memory, Handle 사용량), 서비스정보(동접자 수, 배틀유저 수 등), 서버 상태 정보(업데이트 타임, 데이터베이스 쿼리 타임, 버퍼 사용상태)를 보낸다. 또한 관제툴하고 연결이 되어 있으면서 서버의 상태를 관제툴로 전송하거나, 관제툴의 커맨드를 각 서버로 전송하기도 한다. 관제툴의 커맨드는 서버 시작, 서버 종료, 서버 공지, 서버 서비스 임시 중단, 각 기능에 대한 명령이다. 컨트롤 서버는 주기적으로 각 서버에서 받은 값을 파싱해서 문제점을 파악하고 서버에 문제가 있으면 로그를 남기고, 접속된 관제툴에 메시지를 전송한다. <그림 5>는 컨트롤 서버의 연결도를 나타낸다. 데이터베이스 서버(Database Server)는 각 로직 서버군의 서버들과 연결이 되어서 각 서버의 역할에 맞는 정보를 저장하는 기능을 가지고 있다. Database 저장 부분은 일반적인 게임 서버와 차이가 없다. 다만 통합서버에서는 각 서버에서 처리하는 잡(Job)과 관련된 Database를 연결해서 사용한다. <그림 6>과 같이 Database를 사용하는 서버는 인증(Auth), 클랜(Clan), 커뮤니티(Community), 링크(Link) 4개가 있다.



<그림 6> 데이터베이스 서버 연결도

각 Database의 세부 저장내용은 아래 <표 2>에 나와 있다.

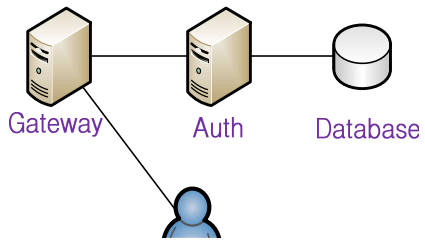
<표 2> Database 세부 정보 표

서버	Database 저장내용
인증 (Auth)	• UserInfo(아이디, 닉네임, 경험치, ETC) • User Inventory, Shop • User Option
클랜(Clan)	• Clan Info
커뮤니티 (Community)	• 쪽지(Note) • 친구(Friend)
링크(Link)	• 옵션(Option): 그래픽, 사운드, 채팅, ETC

3.2.2 게임 시스템 설계

일반적으로 게임에 접속한 클라이언트는 로그인(인증)을 한 후, 로비(마을)에서 인벤토리 정리나, 상점에서 아이템 구입, 클랜, 친구 찾기, 퀘스트, 클라이언트 검색, 등을 한다. 이러한 일련의 준비작업이 끝난 유저는 게임(배틀)룸 리스트를 보고 게임을 할 방을 검색 후 룸에 들어가서 게임(배틀)을 하게 된다. 이렇게 클라이언트가 게임을 하기 위해서 기본이 되는 4가지 시스템(Authentication System-인증 시스템, Lobby System-로비(마을) 시스템, Match Making System-매치메이킹 시스템, Battle System-배틀 시스템)이 있다.

Authentication System(인증 시스템)은 유저가 게임을 하기 위해서 아이디와 패스워드를 가지고 로그인하는 시스템이다. 게이트웨이, 인증, 데이터베이스 서버들이 연결되어서 인증을 처리하다. 게이트워이서버는 연결단을 처리하고 인증서버는 인증시스템을 담당하며, 데이터베이스서버는 ID와 PASSWORD의 정보값 저장을 처리한다.



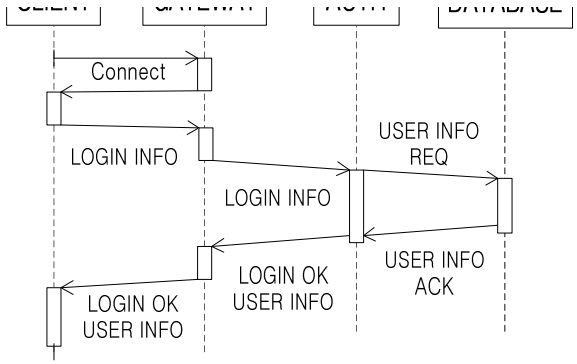
<그림 7> 인증 시스템 연결도

인증을 시도할 때 클라이언트와 서버간 주고받는 데이터 값은 <표 3>과 같다.

<표 3> 인증 시 클라이언트-서버간 데이터

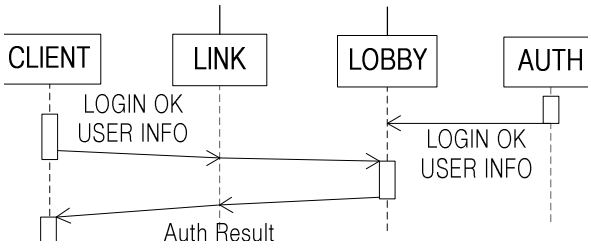
```
STRUCTURE
{
char pId[STRING_SIZE]
char pPass[STRING_SIZE]
}STRUC_LOGIN_INPUT
로그인 요청값(A)

STRUCTURE
{
 INT32  iReturnValue;
 UINT32 iLinkServerIdx;
 UINT32 iLobbyServerIdx;
 UINT32 iUserIdx;
}STRUC_LOGIN_OUTPUT
로그인 결과값(B)
```

위의 구조체(스트럭쳐) 값을 가지고 클라이언트는 서버에게 로그인을 요청하게 된다. 로그인할 때 순서도는 <그림 8>과 같다.



<그림 8> 인증 시스템 흐름도

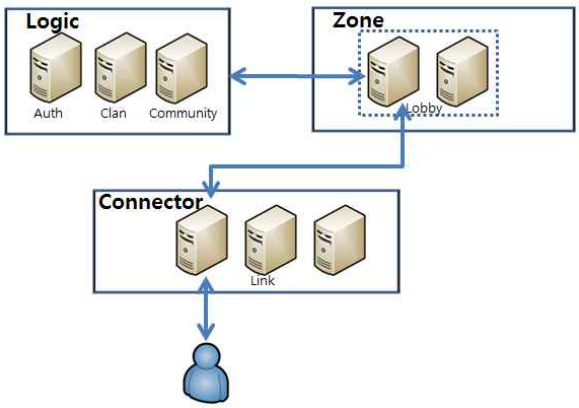
먼저 게이트웨이 서버에 접속한 클라이언트는 아이디와 패스워드를 가지고 서버에 인증 절차를 받게 된다. 게이트웨이 서버는 유저에게 받은 정보(A : STRUC_LOGIN_INPUT)를 인증서버에 보내고. 그 정보를 받은 인증 서버는 데이터베이스에 있는 아이디, 패스워드 정보와 비교하여 로그인 결과 값을 게이트웨이서버를 통해서 클라이언트에게 전달한다. 인증 이후 로그인이 성공된 유저라면 인증서버는 각 로비서버, 링크서버의 클라이언트수를 확인하고 클라이언트가 접속을 해야 하는 서버의 인덱스를 인증성공 정보(B : STRUC_LOGIN_OUTPUT)와 함께 게이트웨이서버에 전달하고 게이트웨이서버는 클라이언트에게 전송한다. 인증성공 패킷을 받은 클라이언트는 받은정보(B : STRUC_LOGIN_OUTPUT)에서 접속해야할 링크서버로 접속을 시도한다.



<그림 9> 로그인 성공 후 배정받은 서버접속 흐름도

링크서버에 접속한 유저는 게이트웨이 서버로부터 받은정보(B : STRUC_LOGIN_OUTPUT)를 링크서버에 보내고 링크서버는 이 정보를 가지고 배정받은 로비서버로 클라이언트 접속 허가 요청을 보낸다. 로비서버는 링크서버로 받은 정보(UID: USER INDEX)값을 가지고 인증서버로부터 이 값이 유효한 값인지를 체크하고 유효한 값이라면 클라이언트에게 로그인 성공 패킷을 보내서 최종 접속 결과를 전송한다. 만약 이 값이 유효하지 않다면 게임서버는 로그에 “비정상 유저” 로그를 남기고 링크서버에 클라이언트에게 디스컨넥션 패킷을 보내고, 링크서버가 클라이언트와 연결을 끊는다.

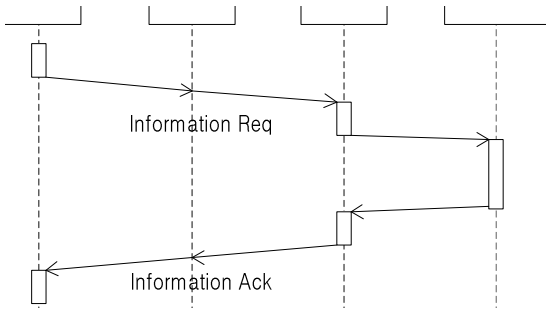
Lobby System(Lobby Server, 로비(마을) 시스템)은 인증(로그인)이 끝난 클라이언트가 게임(배틀)전까지 머무는 장소이다. 로비시스템은 클라이언트들에게 배틀 이외에 여러 가지 정보를 제공한다. 로비시스템이 제공하는 기능은 클랜 정보, 커뮤니티 정보, 상점 &



<그림 10> 클랜, 커뮤니티, 상점 정보 연결도

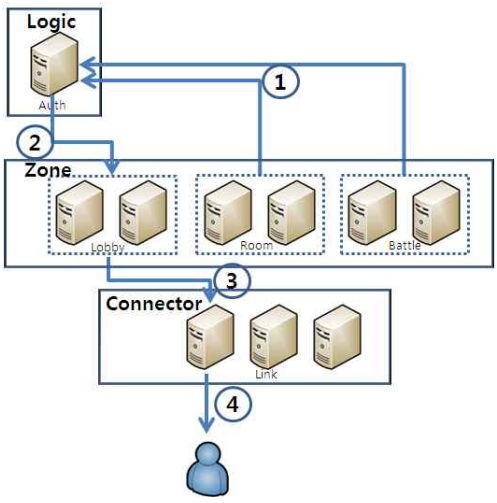
인벤토리 정보, 배틀룸 리스트 정보, 채팅, 퀘스트이다. 클랜 정보와 커뮤니티 정보, 상점&인벤토리 정보, 퀘스트의 기능은 모든 로직서버군이 처리를 하는 기능이다. 클랜정보는 클랜서버에서, 커뮤니티정보는 커뮤니티서버에서, 상점, 인벤토리, 퀘스트 정보는 인증서버에서 처리하게 된다.

<그림 10>과 같이 로비시스템은 클라이언트가 로직서버에게 일련의 일(Job)을 요청하고 로직서버가 요청된 일을 처리해서 클라이언트에게 전달하는 방식으로 처리된다. <그림 11>과 같이 링크서버는 패킷을 바이패스하고, 로비서버는 패킷을 파싱해서 전달할 클라이언트를 정해서 전달하게 된다.

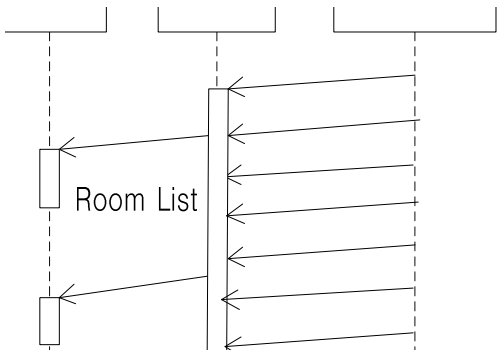


<그림 11> 클랜, 커뮤니티, 상점, 인벤토리, 퀘스트 정보 흐름도

배틀룸 리스트 기능은 <그림 12>와 같이 로비시스템에서 매우 중요한 역할인 배틀룸 리스트를 클라이언트에게 전달하는 기능이다. 이 기능을 처리하기 위해서 인증(Auth)서버와 지역(Zone)서버군의 모든 서버, 그리고 링크(Link)서버가 연동하게 된다. <그림 13>과 같이 룸(Room)서버와 배틀(Battle)서버는 룸 정보를 인증(Auth)서버로 전달하고 인증서버는 받은 룸 리스트를 합쳐서 로비(Lobby)서버로 전달하고, 로비(Lobby)서버는 링크(Link)서버를 통해서 룸 정보를 클라이언트에게 전달하게 된다.

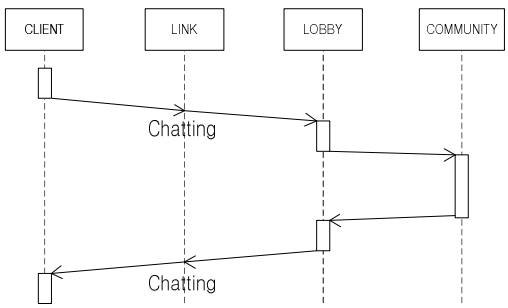


<그림 12> 배틀룸 리스트 정보 연결도



<그림 13> 룸 리스트 정보 흐름도

룸, 배틀(Battle)서버는 일정시간(500ms)마다 처리하고 있는 룸 정보를 인증서버로 보낸다. 인증서버는 모든 룸, 배틀 서버의 정보를 모아서 일정시간(1,000ms)마다 각각의 정보를 로비서버로 전달하게 된다. 채팅시스템은 로비서버와 커뮤니티서버를 통해서 각 클라이언트들에게 전달된다.



<그림 14> 채팅 정보 흐름도

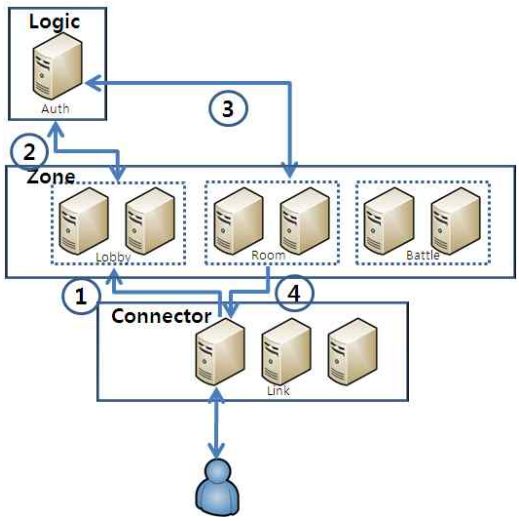
채팅시스템은 매우 간단한 플로우로 처리가 된다. 링크, 로비서버는 채팅 내용을 커뮤니티 서버에게 전달하고 커뮤니티 서버가 각 서버로 전송하는 역할을 한다.

Matchmaking System(Lobby, Room, Battle Server-매치메이킹시스템)은 클라이언트들이 게임을 할 수 있도록 연결해 주는 역할을 한다. 매칭시스템에

관련된 서버들은 로비서버, 룸 서버, 배틀 서버가 로직 서버군의 인증 서버를 통해서 분산처리를 하면서 방을 생성하고 유저를 이동시키는 작업을 하게 된다. 이 시스템은 기본적으로 4가지의 상태변화(게임룸 생성-로비 서버에서 룸 서버로 이동, 게임룸에서 배틀룸으로 상태변화-룸 서버에서 배틀 서버로 이동, 배틀룸에서 게임룸으로 상태변화-배틀 서버에서 룸 서버로 이동, 게임룸 종료-룸 서버에서 로비 서버로 이동)를 가지고 있으며, 각 상태 변화를 하면서 클라이언트는 지역서버군의 3개의 서버를 이동하게 된다. 위와 같이 4가지 상태변화를 하면서 클라이언트는 로비, 룸, 배틀 서버로 이동하게 되고, 로직 서버군에 인증서버는 상태변화를 인증해주는 역할을 한다.

· 게임룸 생성

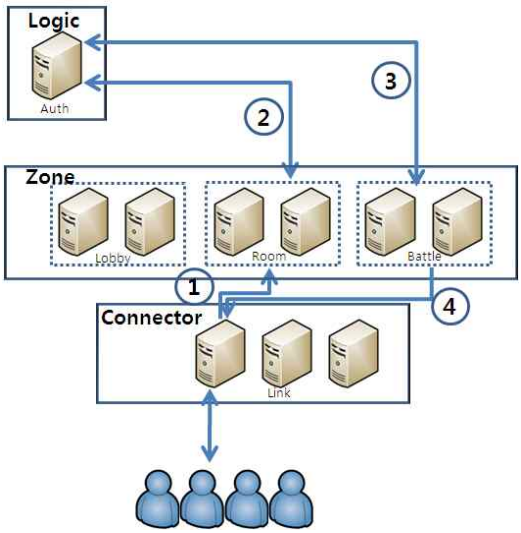
<그림 15>은 로비에 있는 클라이언트가 룸으로 이동하는 모습을 보여주고 있다. 로비에 있는 클라이언트는 로비서버를 통해서 인증서버에 룸으로 이동하겠다는 메시지를 보내고 그 메시지를 받은 인증서버는 이 요청이 가능한지 룸서버에 확인하고, 가능하면 로비서버, 불가능하면 룸서버를 통해서 클라이언트에게 결과값을 통보한다. 성공적으로 이동이 되면 클라이언트의 정보는 룸서버로 이동하게 된다.



<그림 15> 로비에서 룸으로 이동

· 게임룸에서 배틀룸으로 상태변화

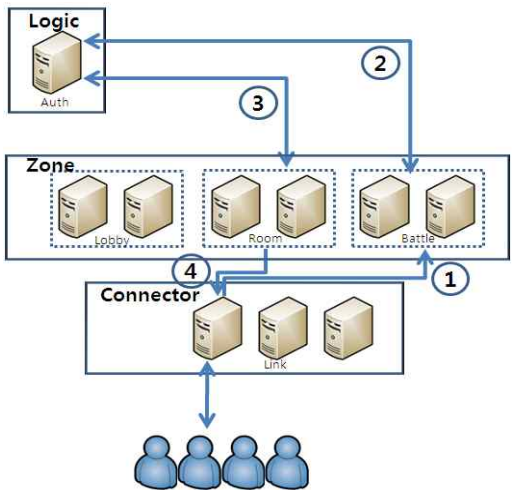
룸 서버에 게임할 수 있는 클라이언트들 그룹(게임방)이 배틀할 준비가 되면 룸서버에 있는 그룹(게임방)에서 최종으로 게임시작을 룸서버에 요청하고 요청을 받은 룸서버는 인증서버로 배틀룸을 요청하게 된다. 배틀룸 생성이 가능하면, 인증서버는 룸에 있는 모든 유저를 배틀서버로 이동하는 일련의 작업을 하게 된다.



<그림 16> 룸 서버에서 배틀서버로 이동

· 배틀룸에서 게임룸으로 상태변화

배틀이 끝난 룸은 배틀서버에서 삭제가 되고 배틀룸에 있었던 클라이언트들은 룸서버로 이동하게 된다. “게임룸에서 배틀룸으로 상태변화” 역순으로 동작한다.



<그림 17> 배틀서버에서 룸서버로 이동

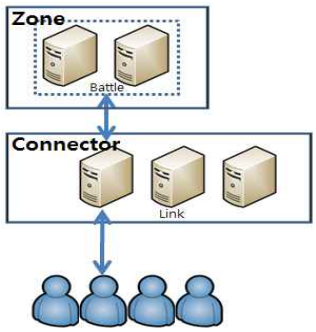
· 게임룸 종료

게임룸으로 이동한 클라이언트는 룸은 다시 배틀서버로 이동해서 배틀을 하거나, 로비로 돌아갈 수 있다. 다시 배틀을 하면 “게임룸에서 배틀룸으로 상태변화”를 통해서 다시 배틀이 가능하다. 반대로 룸에서 나온 클라이언트는 로비서버로 이동하게 된다

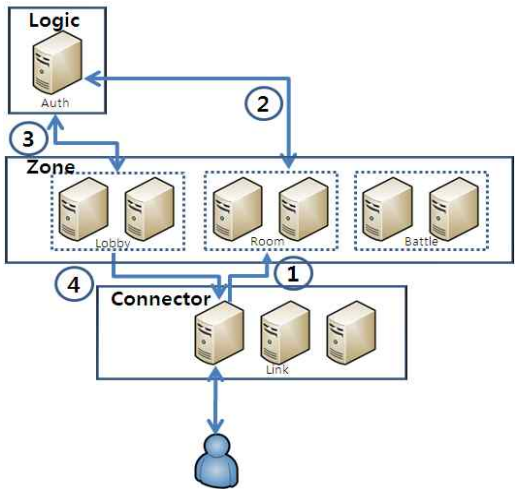
· 배틀 시스템(Battle System)

배틀 시스템은 게임시스템의 매치메이킹이 끝난 클라이언트들이 같이 게임을 할 수 있도록 해주는 시스템이다. 배틀 시스템은 유저들의 배틀상태에 대한 모든 작업을 담당하며, 배틀 종료시. 게임의 전적 및 포인트를 저장하는 기능도 같이 하게 된다. 이러한 일들을 처리하기 위해서 배틀 서버는 클라이언트의 싱크처리, 월드 컬리젼, NPC 컬리젼, AI, 포인트 계산 등을 처리해야 한다. 이러한 작업은 많은 CPU, 메모리, I/O부하를 필요로 한다.

<그림 19>와 같이 배틀시스템은 모든 일(Job)처리는 배틀(Battle)서버에서 처리하고 정보전달은 링크서버를 통해서 하게 된다. 게임 장르마다 조금씩 다르겠지만 배틀서버는 초당 10~15회 정도의 배틀 패킷을 클라이언트에 전달해야만 렉(Lack)없는 게임이 가능하다.



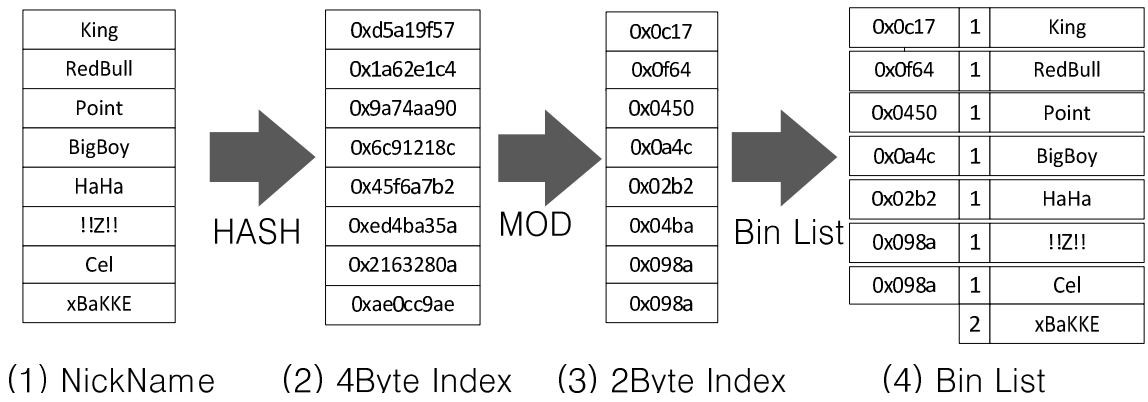
<그림 19> 배틀 연결도



<그림 18> 룸서버에서 로비서버로 이동

3.3 속도 향상 기법

게임서버에서 속도에 가장 큰 영향을 미치는 두 가지 요인은 첫 번째로 대규모 데이터에서 특정정보를 찾는 검색하는 방법, 두 번째로는 쓰레드(Thread)에서 동시 접근하는 버퍼(Buffer)의 동기화 방법이다. 이 두 가지 요인 은 게임서버가 동작할 때 가장 많이 사용된다. 그러므로 이 요인을 빠르게 만들면 좋은 성능을 나타내는 서버개발이 가능하다. 아래 2개의 기법은 검색, 동기화를 좀 더 빠르게 해주는 기법이다.



<그림 20> 데이터 변환

3.3.1 변형된 해쉬 기법

게임서버는 수많은 클라이언트들이 접속해서 동시에 게임을 가능하게 해주는 역할을 한다. 이러한 게임의 역할에 기반이 되는 것이 커뮤니티 기능이다. 커뮤니티의 기능은 클라이언트간의 친구, 클랜, 쪽지, 채팅 기능을 의미한다. 이러한 기능을 제공해주기 위해서는 대규모 데이터를 관리 및 검색 기능이 매우 중요하다. 통합서버에서는 해쉬 함수와 바이너리리스트를 사용해서 이러한 기능을 빠르게 제공한다. <그림 20>은 게임에서 사용하는 스트링 데이터(닉네임 or ID)를 빨리 검색하기위해서 데이터를 변환 과정이다. 첫 번째로 데이터를 해쉬기법을 사용해서 4바이트의 특정한 정수값으로 변환한다. 이 변환을 위해서 간단한 해쉬 변환식을 사용했다. [(1)에서 (2)로 이동]

key=strString[i]+(key<<6)+(key<<16) - key [SDBM hash function 사용]

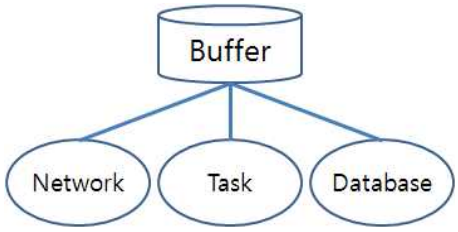
두 번째로 변환된 4바이트 정수값을 특정 숫자보다 작게 하기 위해서 모듈라연산(%)을 사용한다. [(2)에서 (3)로 이동]

Key=Key%4000 [데이터(2)<N] N = 4000

세 번째로 이렇게 나온 2바이트 정수값은 동일한 정수값이 나올 수 있다. 이러한 문제를 해결하기 위해서 바이너리리스트를 사용해서 한 저장소에 넣는다. [(3)에서 (4)로 이동]

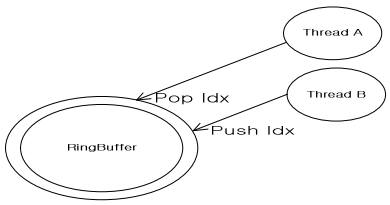
3.3.2 쓰레드 링버퍼 기법

게임서버에서 2개의 쓰레드간 버퍼의 동기화가 필요한 경우가 많이 발생한다. 보편적인 방법은 임계영역(Critical Section)을 사용해서 버퍼를 보호하는 방법을 사용하지만, 너무 많은 크리티컬섹션은 서버시스템에 병목현상을 생기게 하고, 서버개발자가 파악하기도 힘든 경우가 많다. 이러한 문제점을 피하기 위해서 링퍼버 기법을 제안한다. 링버퍼 기법은 2개의 쓰레드의 무결성 데이터통신을 위해서 사용한다. 일반적인 동기화 기법을 사용하는 게임 서버의 기본 프로그래밍 구조는 <그림 21>과 같다.



<그림 21> 서버 프로그램 구조

각 모듈(Network, Task, Database)간의 하나의 버퍼를 공유하기 위해서 프로그램에서는 1개의 임계영역을 사용하게 된다. 서버는 평균적으로 10개 이상의 쓰레드를 사용하게 된다. 쓰레드를 사용하면서 동기화를 하기 위해서 많은 쓰레드들이 임계영역을 사용해야 한다. 하지만 무분별하게 사용한 동기화 기법은 속도를 저하 시키거나 데드락이 발생할 수 있다. 이러한 문제를 해결하기 위해서 링 버퍼를 사용해서 2개의 쓰레드간의 동기화를 해결한다. 버퍼전체에 임계영역을 사용하는 기존방식 대신 4byte 변수인 PopIdx, PushIdx에만 인터락(interlock)을 사용하는 방식으로 버퍼락이 걸리는 시간을 최소한으로 줄였다.



<그림 22> 쓰레드 링버퍼

위의 <그림 22>에서 쓰레드 B 는 데이터를 전송하는 쓰레드이고, 쓰레드 A는 데이터를 받는 쓰레드이다.

각 쓰레드는 변수에 대해서 아래와 같은 권한을 가지고 있다.

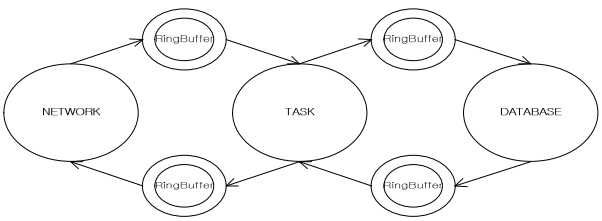
<표 4> 쓰레드 변수 권한

목록	PushIdx	PopIdx
쓰레드 A	Read	Write / Read
쓰레드 B	Write / Read	Read

```
INT32 Push(void * pBuffer, INT32 i32Size)
//데이터 넣기
{
//버퍼의 인덱스를 검사한다.
     if(m_iPopIdx > BUFFER_MAX_COUNT)return
        ERROR_BUFFER_FULLIDX;
     if((m_iPushIdx-m_iPopIdx)>=m_iBufferCount)
        return ERROR_BUFFER_FULL;

// 데이터를 카피한다.
     INT32 iWriteIdx=m_iPushIdx%m_iBufferCount;
     MemCopy(&m_pBuffer[iWriteIdx*
          m_ui32BufferSize], pBuffer, i32Size);
     m_iPushIdx++;

//인덱스를 정리한다.
     if( m_iPushIdx > m_ui32WriteMax )
          m_ui32WriteMax = m_iPushIdx;
     return SUCCESS;
}
```

```
VOID * Pop(void)
// 데이터 가지고 오기
{
     void * pBuffer = NULL;
//가지고 올 수 있는 데이터가 있는지 확인한다.
     if( 0 < GetUseBufferCount() )
     {
//데이터의 포인터를 찾아서 전달한다.
          INT32 iReadIdx = m_iPopIdx %
                                   m_iBufferCount;
          pBuffer = &m_pBuffer[iReadIdx *
                                   m_ui32BufferSize];
     }
     return pBuffer;
}
```

링 버퍼 기법을 사용해서 동기화를 하게 되면 서버 프로그램 구조가 아래와 같이 변경된다.



<그림 23> 쓰레드 링버퍼를 사용한 서버 프로그램 구조

<그림 23>과 같이 각각의 쓰레드는 4개의 링버퍼에 각 하나씩의 권한을 가지게 된다. 예를 들어, 네트워크 쓰레드는 왼쪽 상단에 있는 링버퍼에 쓰기 권한을 가지면서 왼쪽 하단의 링버퍼에는 읽기 권한을 가지게 된다.

링버퍼 기법은 쓰레드간 한 자원을 차지하기 위한 불필요한 시간(Lock, unlock)을 줄일 수 있지만, Pop하는 쓰레드가 Push하는 쓰레드보다 속다가 느리면 버퍼가 손실, 링버퍼가 필요(메모리 오버헤드 발생)한 단점이 있다. 2가지 단점이 영향이 없는 모듈에서 사용한다면 아주 빠른 결과를 낼 수 있는 모듈개발이 가능하다.

4. 통합 서버 시스템 구현 및 성능 평가

본 장에서는 3장에서 설계한 통합 서버를 구현하고 이 시스템의 성능을 평가한다. 통합 서버는 기존 서버의 설계보다 처리 속도가 향상된 서버를 만드는 것이 아니라, 경계 없는 가상서버를 만드는 것이다. 그렇기 때문에 이 서비스 평가에서는 구현된 통합 서버가 일반적인 게임서버방식과 비교하여 서비스품질이 떨어지지 않는 것을 증명하도록 한다.

4.1 시스템 구현

서버의 하드웨어는 현재 게임서버들의 평균 사양인 Intel Xeon E5620 CPU와 메모리는 4GByte 스펙으로 설정한다. 외부 처리를 위한 서버(릴레이서버)에는 네트워크 분산처리용으로 네트워크카드를 기본 2개에서 4개를 추가해서 총 6개를 사용하고, 내부는 외부용과 동일한 사양에 내부, 외부 네트워크카드를 2개를 사용한다. 테스트 서버에서 가장 부하가 심한 데이터베이스는 CPU를 하나 더 추가해서 16개 코어를 만들고, 메모리도 16GByte를 장착, 그리고 하드디스크 설정은 RAID5로 하였다. 세부 스펙은 아래와 같다.

<표 5> 물리서버 스펙-연결 서버군

목록	사양	개수
CPU	Intel Xeon E5620	1
Board	INTEL S5520HC	1
RAM	4G	4
HARD DISK	500G	1
Network Card	EXPI9301CT	2 + 4
OS	Windows 2003 Server	1

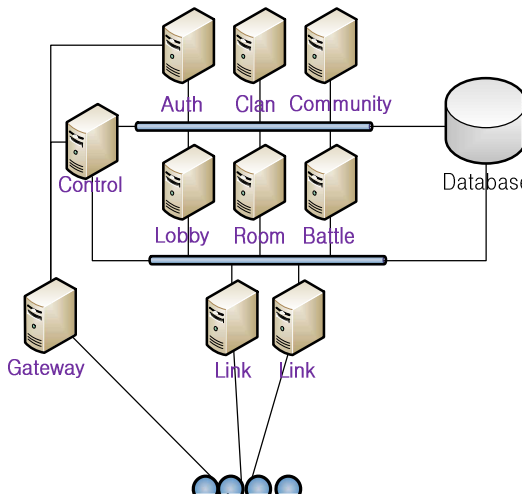
<표 6> 물리서버 스펙-로직서버군, 지역서버군, 어시스트 서버군

목록	사양	개수
CPU	Intel Xeon E5620	1
Board	INTEL S5520HC	1
RAM	4G	4
HARD DISK	500G	1
Network Card	내장(INTEL S5520HC)	2
OS	Windows 2003 Server	1

<표 7> 물리서버 스펙-내부 분산처리 서버

목록	사양	개수
CPU	Intel Xeon E5620	2
Board	INTEL S5520HC	1
RAM	2G	8
HARD DISK	1T (RAID5)	5
Network Card	내장(INTEL S5520HC)	2
OS	Windows 2003 Server	1
DB	MS Database	1

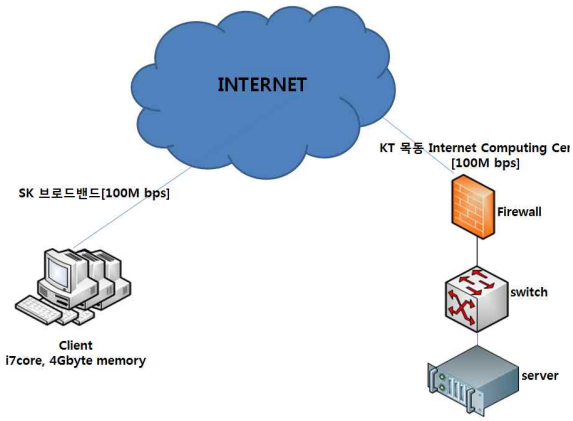
테스트 환경은 <그림 24>같은 환경으로 구축하였다. 연결서버군(Connection)은 한 개의 게이트웨이 서버와 두 개의 링크서버를 가지고 클라이언트와 연결을 구성하였고, 그 외 지역서버군(Zone), 로직서버군(Logic), 어시스트서버군(Assist)은 각각 하나씩 서버를 구성했다.



<그림 24> 테스트 환경 구축도

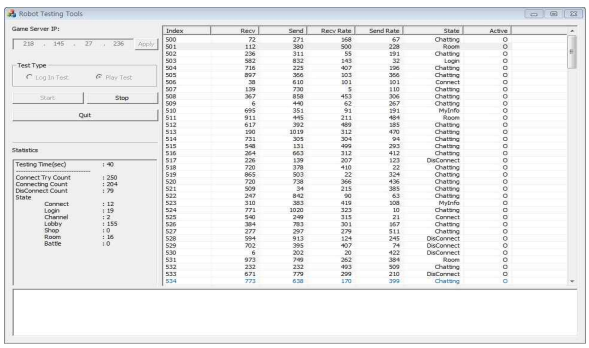
4.2 성능 평가

통합 서버는 불필요한 유저이동을 없애기 위한 목적으로 개발 되었다. 여기서는 기본적인 게임서버 서비스품질(Qos)에 만족하는지를 평가한다.



<그림 25> 테스트 네트워크 환경

테스트를 위해서 <그림 25>와 같은 네트워크 환경을 만들고, <그림 26>과 같은 간단한 "테스트 클라이언트" 툴을 개발해서 테스트를 하였다. 이 툴은 2,000개의 가상 클라이언트 생성이 가능하다.



<그림 26> 테스트 클라이언트

테스트 클라이언트는 다음과 같은 절차로 진행된다.

① 자동으로 로그인, 로비, 룸, 배틀 상태로 이동하여 일반 클라이언트처럼 동작한다.

② 각 클라이언트의 Send, Recv 패킷을 표시한다.

③ 각 클라이언트의 로그인, 로비, 룸, 채팅, 배틀 등과 같은 상태를 표시한다.

④ 전체 클라이언트의 로그인, 개별 클라이언트 상

태와 같은 통계 표시한다.

테스트를 위해서 PC 5대에 테스트 클라이언트를 설치하여 테스트를 진행했다.

<표 8> 통합서버 응답 시간 비교 분석

	CCU 5,000	CCU 10,000
LogIn	60ms	61ms
Clan	59ms	62ms
Community	59ms	61ms
Create Room	55ms	55ms
Start Battle	55ms	55ms
Battle	52ms	52ms

<표 8>에서 나온 결과와 같이 동접이 늘면서 커뮤니티와 클랜의 응답 속도는 2ms 정도의 오버헤드가 발생하였지만 클라이언트가 체감할 수 없는 속도차이를 보였다.

<표 9> 일반서버 응답 속도 비교 분석

	CCU 5,000	CCU 10,000
LogIn	60ms	61ms
Clan	56ms	59ms
Community	56ms	59ms
Create Room	53ms	53ms
Start Battle	52ms	52ms
Battle	50ms	51ms

일반 게임 서버 보다 통합서버는 평균 3ms 정도 늦었지만, 3ms는 클라이언트가 느끼지 못하는 수준의 시간 단위이다. 3ms는 Link Server를 사용하면서 늘어난 오버헤드이다. 위의 응답속도 표를 보면 알 수 있듯이 통합서버는 클라이언트가 요청한 모든 요청을 별다른 속도지연 없이 처리할 수 있음을 알 수 있다.

<표 10> 동접 대비 하드웨어시스템 비교 분석

	CCU 5,000	CCU 10,000
CPU(%)	20	34
Memory(MByte)	1,100	1,700
Handle(Count)	23000	28000
Bandwidth(Mcps) [Public Network]	Up:30 Down:200	Up:60 Down:500

하드웨어의 성능은 동접이 비교 평균값은 별다른 차이가 없을 볼 수 있다. TCP연결이 많아지면서 각 수치들이 1.5배에서 2배 사이로 증가한 것을 볼 수 있다.

<표 8>과 <표 10>의 결과값을 가지고 50,000명, 100,000명일때의 필요 물리서버수를 예측해 봤다.

<표 11> 동시 접속자 대비 필요 물리 서버

	CCU 5,000	CCU 10,000	CCU 50,000	CCU 100,000
Connector	2	2	6	11
Zone	3	3	3	6
Logic	3	3	3	3
Assist	2	2	3	4

통합서버는 <표 12>와 같이 3가지 온라인게임의 품질요구 조건인 확장성, 일관성, 최소지연성을 만족하는 결과가 나왔다.

<표 12> 품질 요구 조건표

	통합 서버
확장성(scalable architecture)	<표 9>로 확인
일관성(Consistency)	<표 7>,<표 8>로 확인
최소지연성(Latency)	<표 6>으로 확인

5. 결론 및 연구 방향

현재 게임서비스의 많은 부분에서 통합서버 환경이 필수 요소로 대두 되고 있다. 일반적인 유저들은 CBT(Close Beta Service), OBT(Open Beta Service)에 게임클라이언트를 통해 게임서버로 접속하고 기본적인 게임시스템과 서버의 안정성을 확인한다. 실제로 게임서버의 안정성은 게임 퀄리티를 판단하는 큰 잣대가 되고 있다. 캐쥬얼 온라인 게임이 처음 출시되었을 때 유저들은 많은 사람들이 같이 게임을 할 수 있는 환경에 너무 놀랐고, 특히 렉이 없는 환경에서 게임을 할 수 있다는 것에 많이 즐거워했다. 게임적인 요소로 보면 온라인 게임보다 일반 1인용 게임이 더 좋은 게임 환경을 가지고 있다. 하지만 이제 게임 유저들은 혼자 하는 게임보다는 같이 하는 게임을 원하고 같이할 수 있는 온라인 게임중에서 안정적이고, 편

리한 환경을 제공해주는 게임을 원하고 있다. 따라서 게임서버 개발자들은 이러한 유저의 요구에 대응할 수 있도록 편의성을 제공해주어야 한다.

본 논문에서는 많은 유저들이 같은 공간에서 게임을 하는 것처럼 보이게 하는 가상화된 큰 공간서버를 만들었다. 실제 유저들은 각기 다른 물리적인 서버에 접속해 있지만 접속해 있는 모든 유저들은 모두 다 같은 서버에서 게임을 하고 있다고 생각할 것이다. 이 통합서버 시스템을 통해서 유저들은 불필요한 서버이동을 할 필요가 없다. 이 시스템의 최대 장점은 유저 편의성이 많이 좋아지므로, 좀 더 많은 유저들이 게임에 접속할 수 있도록 유도할 수 있을 것이다. 가상화로 응답속도가 저하된 부분을 커버하기 위해서 "변형 해쉬기법"과 "쓰레드 링 버퍼기법"을 제한하였다.(제 3장) 두 개의 성능 향상 기법은 기존의 일반적인 검색, 동기화기법보다 빠른 성능을 보였다.

향후 연구 과제로는 통합 서버에서 처리할 수 있는 최대 유저수를 올려보는 방법에 대한 기술 개발이 필요하다. 즉 내부 네트워크의 부하를 줄일 수 있는 알고리즘, 내부 서버들의 응답속도에 대한 최적화, 각 파트 서버들의 알맞은 부하 분산, 그리고 서버 안정성을 추가로 연구를 해야 할 과제이다.

참 고 문 헌

[1] 권주현, "온라인 게임 서버를 위한 성능 관리 시스템의 설계 및 구현", 단국대학교 정보통신 대학원 석사학위 논문, 2010.

[2] 박진환, "캐쥬얼 게임 시장의 현황과 전망", 정보과학회지, 제23권, 제6호, pp.58-63, 2005.

[3] 김태수, "P2P 구조를 응용한 효율적인 온라인 분산 게임서버", 성균관대학교 정보통신대학원 석사학위 논문, 2009.

[4] 문성원, "분산 Seamless 게임서버에서의 효율적인 게임공간 관리기술", 서강대학교 정보통신대학원 석사학위논문, 2005.

[5] 배유미, 정성재, "정보 보안을 위한 데스크탑 가상화 기술 동향", 보안공학연구논문지, 제8권, 제2호, pp.255-264, 2011.

[6] 송승민, "효율적 IDC를 위한 서버 가상화 구현", 인천대학교 정보통신대학원 석사학위논문, 2010.

[7] 김진미, 안창원, 정영우, 박종근, 고광원, 변일수, 우영춘, "차세대 컴퓨티을 위한 가상화 기술" 전자통신동향분석, 제23권, 제4호, pp.102-114, 2008.

[8] 유석종, "분산게임서버와 공간분할기법", 한국정보과학회, 제23권, 제6호, pp.29-35, 2005.

[9] 이남재, "온라인 롤플레잉 게임을 위한 Full 3D 맵 관리 방법에 관한 연구", 전북대학교 대학원 박사학위논문, 2003.

[10] 장수민, 유재수, "MMORPG 서버의 부하균등화를 위한 효율적인 분산처리 기법", 한국콘텐츠학회논문지, 제7권, 제11호, pp.69-75, 2007.

[11] 양광호, 심광현, 고동일, 박일규, 김종성, "온라인 게임 서버의 기술 동향", 전자통신동향분석, 제16권, 제4호, pp.14-22, 2001.

[12] 정상진, 신명기, 김형준, "미래인터넷을 위한 네트워크 가상화 표준기술 개발 동향", TTA Journal No.132, pp.84-92, 2010.

[13] 나문성, 김승훈, 이재동, "클라우드 환경에서 대규모 콘텐츠를 위한 효율적인 자원처리 기법", 한국산업정보학회논문지, V.15, No.4, pp.17-27, 2010

[14] Jae-Dong Lee, Jin-Sung Kim, "ACSA:An Adaptive Content System Architecture", 한국산업정보학회논문지, V.16, No.2, 2011.

김 성 백 (SungBaek Kim)

- 준회원
- 단국대학교 전자계산학과 학사
- 단국대학교 정보컴퓨터과학과석사

- 관심분야 : 온라인 게임 서버, AI, 해킹방어

이 재 동 (Jae-Dong Lee)

- 정회원
- 인하대학교 전자계산학과 학사
- Cleveland State University 석사
- Kent State University 박사

- 단국대학교 공과대학 소프트웨어학과 교수(현)
- 단국대학교 국제처장(현)
- 단국-삼성 모바일연구소 소장(현)
- 관심분야 : Mobile Technologies/Applications, Contents Technologies, IT Strategy

논 문 접 수 일 : 2012년 03월 13일
1차수정완료일 : 2012년 04월 26일
2차수정완료일 : 2012년 05월 07일
게 재 확 정 일 : 2012년 06월 15일