Journal of Korea Game Society 2016 Jun; 16(3): 97-108
http://dx.doi.org/10.7583/JKGS.2016.16.3.97

게임 프로그래밍

클라이언트-서버간 거리 편차의 최소화를 위한 클라이언트의 서버 배정 방법

이성해, 김상철
한국외국어대학교 컴퓨터 및 전자시스템 공학부
liii1117@daum.net, kimsa@hufs.ac.kr

A Method for Assigning Clients to Servers for the Minimization of Client-Server Distance Deviation

Sunghae Lee, Sangchul Kim
Computer Science and Engineering Major, Graduate School,
Hankuk University of Foreign Studies

요 약

다수 클라이언트(사용자)들이 동시에 진행하는 온라인 게임은 대부분 다중 서버 구조를 채택하고 있다. 그룹 플레이의 경우, 같은 그룹내 클라이언트들 사이에 사용자 반응시간에 큰 차이가 나면 게임의 공정성과 흥미를 떨어뜨리게 된다. 본 논문에서는 신규 클라이언트들을 대상으로, 이런 시간의 중요한 요소인 클라이언트-서버간 거리의 편차를 최소하도록 클라이언트를 서버에 배정하는 방법을 제안한다. 본 방법은 그룹 플레이를 위한 클라이언트 매칭과 서버 부하 균등도 함께 지원하고 있다. 우리는 클라이언트-서버 배정 문제를 IP(Integer Programming)으로 모델링라고 그 해를 구하는 GA(Genetic Algorithm) 기반의 알고리즘을 제안한다. 우리는 본 논문에서 재안한 방법을 다양한 조건하에서 실험했고 그 특성을 분석한다. 우리가 조사해 본 바, 클라이언트 매칭과 서버 부하를 고려하면서, 클라이언트-서버 거리 편차를 최소화하는 클라이언트-서버 배정에 관한 기존 연구는 많지 않았다.

Keywords : 클라이언트-서버 배정, 게임 서버, 유전자 알고리즘, 서버 부하 균등

ABSTRACT

Multi-client online games usually employ multi-serve architectures. For group play, if the user response time deviation between the clients in a group is large, the fairness and attractions of the game will be degraded. In this paper, given new clients, we propose a method for assigning the clients to servers to minimize the deviation of client-server distance which plays a major role in the user response time. This method also supports client matching for group play and server load balancing. We formulate the client-server assignment problem as an IP one, and present a GA(Genetic Algorithm)-based algorithm to solve it. We experimented our method under various settings and analyzed its features. To our survey, little research has been previously performed on client-server assignment under consideration of client matching, distance deviation minimization and server load balancing.

Keywords : Client-Server Assignment, Game Servers, GA, Server Load Balancing

Received: May, 13, 2016 Revised: Jun, 15, 2016
Accepted: Jun, 20, 2016
Corresponding Author: Sangchul Kim(Hankuk Universty of Foreign Studies)
E-mail: kimsa@hufs.ac.kr
ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

동시에 접속하는 클라이언트(또는 사용자)가 많은 온라인 게임의 경우에는 원활한 게임 진행을 위해 여러 서버들을 운영하게 된다. 다중 게임 서버 환경에 관한 주요 연구 이슈 중 하나가 클라이언트들을 서버에 배정하는 것이고[1], 이를 클라이언트-서버 배정 (Client-Server Assignment)이라 부른다. 클라이언트-서버 배정에서는 여러 가지 목표 지표들을 최적화하고자 하는데, 본 논문에서는 클라이언트와 서버간의 거리 편차, 서버 부하 균등 및 클라이언트 매칭(client matching)에 초점을 둔다.

게임의 성격상 또는 흥미를 높이기 위해, 많은 온라인 게임들은 일정 수의 클라이언트들이 서로 경쟁하는 형태로 진행된다. 이와 같은 그룹 플레이에서는 사용자 반응 시간의 편차를 줄이는 것이 중요하다. 특히 실시간 게임에서는 이런 편차가 크면 게임 플레이의 공정성과 정확성을 해치게 되어, 결국 사용자들의 게임에 대한 흥미를 떨어드리게 된다. 사용자 반응시간은 크게 클라이언트 서버간의 지연시간과 서버의 부하 처리 시간에 영향을 받는다.

트래픽이 인터넷 상에서 지역적으로 골고루 분산된다고 가정하면, 클라이언트-서버간 거리는 메시지 전달에 걸리는 지연시간(delay)에 큰 영향을 미친다. 따라서 클라이언트-서버 배정에서 클라이언트-서버간 거리 편차의 최소화가 중요한 이슈가 된다. 게임 서버의 부하는 주로 접속한 클라이언트들의 액션에 대한 처리와 관련된 타 서버들과의 통신으로 발생한다[2, 3]. 따라서 서버 부하 균등도 클라이언트 관점에서 보면 사용자 반응 시간의 편차를 줄이는 역할을 한다.

클라이언트 매칭은 그룹 플레이를 위해 클라이언트들을 그룹으로 묶는 것을 말한다. 클라이언트 그룹의 크기는 게임의 성격에 따라 달라지는데, 예를 들어 바둑은 2명, 마작은 4명이다. 온라인 게임에서 클라이언트 매칭은 주로 로비 서버 (lobby server)에서 이루진다. 수동 매칭의 경우, 각 클라이언트는 원하는 상대방들을 직접 찾게 된다. 반면에 자동 매칭의 경우에는 게임 시스템이 적절한 방법으로 클라이언트들을 그룹으로 묶게 된다. 자동 매칭의 예로서, 최근 사행 방지를 위해 국내의 웹보드 게임에서 제공하고 있는 무작위 매칭이다.

클라이언트-서버 배정과 게임 서버의 부하 균등에 관한 기존 연구는 적지 않게 수행되었다. 하지만 클라이언트 매칭을 지원하면서 클라이언트 서버간 거리 편차 최소화 및 서버 부하 균등을 동시에 고려한 클라이언트-서버 배정에 관한 연구는 많이 발표되지 않았다.

본 논문에서 우리는 로비 서버에 새로이 도착한 클라이언트들을 대상으로, 이들을 정해진 그룹 크기만큼 무작위로 묶은 후 서버에 배정하는 방식을 제안한다. 이때 클라이언트-서버간의 거리 편차를 최소화하고, 서버 부하의 균등도 제약조건 형태로 함께 고려한다. 우리는 한 게임 그룹은 동일한 서버에 배정되는 경우를 가정한다. 따라서 서버들간의 수반되는 통신 오버헤드는 크지 않기에 서버간의 부하 차이는 서버에 접속된 클라이언트 수의 차이에 거의 좌우된다고 할 수 있다.

우리는 클라이언트-서버 배정 문제를 IP(Integer Programming)으로 모델링하고, 그 해를 구하는 유전자 알고리즘 (GA: Generic Algorithm)[4] 기반의 휴리스틱 알고리즘을 제시한다. 우리의 클라이언트-서버 배정 문제는 특수한 형태의 bin-packing 문제인데, 이것은 NP-complete 문제이다. GA는 이와 같이 폴리노미얼 타임에 해를 구할 수 없는 어려운 문제를 해결하는 대표적인 기법이다.

다중 서버 환경에서의 클라이언트-서버 배정에 관해 다수의 기존 연구 결과가 발표되었다[1,5,6,7,8,9,10]. [5,6]에서는 클라이언트들이 직접 원하는 게임 서버를 찾을 수 있는 기능을 제공한다. [5]는 FPS 게임 환경 하에서 서버 탐색을 지원하는 에이전트를, [6]은 클라이언트가 지정한 게임 서버 특성에 부합되는 게임 서버를 자동 탐색

하는 방법을 제안했다. 여기서 서버 특성이란 지원하는 게임 맵, 게임 모드, 그룹으로 하는 게임인 경우에 그룹 크기 등이다

앞선 방법은 결국 클라이언트가 수동으로 서버를 선택하는 것으로서 번거로운 작업일 뿐만이 아니라 서버 자원을 효율성으로 활용하는데 한계가 있다. 따라서 대부분의 클라이언트-서버 배정에 관한 기존 연구는 자동 배정에 관한 것으로서, 특정 목표 지표를 최대화 또는 최소화하도록 노력한다. 대표적인 목표 지표로서, 서버-클라이언트간의 평균 지연 시간[7,8], 제약 조건을 만족하는 클라이언트 수[1], 클라이언트간의 최대 지연 시간[9], 주어진 클라이언트 그룹을 배정할 서버의 수[10], 서버 균등 등이 있다. 대부분 기존 연구에서는 목표 지표의 최적화에 유리한 순서대로 차례로 클라이언트들을 위한 서버를 선택해 나가거나[1] 또는 초기 서버 배정 후 반복적으로 배정을 개선해 가게 된다[10]. 이들 목표지수는 서로 독립적이어, 이들 중 한 개를 최적화 대상으로 정하고, 필요한 다른 지표들은 제약 사항들(constraints)로 고려하게 된다.

온라인 게임에서 다수 클라이언트들로 인해 발생하는 부하를 서버들에 균등하게 배분하는 연구도 많이 발표되었다[2,3,11,12,13,14,15]. [11]에서는 게임 월드를 여러 작은 셀로 나누고, 이들을 묶어 존으로 구성하는데, 존들에 속한 클라이언트의 수가 균등하도록 하는 방법을 발표했다. [12]에서는 클라이언트 수와 통신 오버헤드로부터 부하를 결정하는 함수를 이용해, 서버의 최대 부하를 최소화하는 SA(Simulated Annealing) 기반 알고리즘을 제안했다. [2]에서는 과부하가 걸린 존이 발생하면, 서버별 클라이언트 수가 균등해 지도록 게임 월드를 존의 크기를 재조정한다. 앞선 기존 연구에서는 부하 균등을 전담하는 서버가 있다고 가정한다. 하지만 [3,13,14]에서는 과부하된 서버가 자신의 부하의 일부를 주변 서버들에 이전시키고, [15]에서는 서비스 속도가 늦다고 판단하는 사용자가 접속 서버를 다른 것으로 옮기는 분산형 방법을 제안했다. 이와 같은 분산형 모델은 과부하시 빠른 조치가 가능하다는 장점이 있지만 부하 균등의 최적화를 달성하는데 한계가 있다.

온라인 게임에서의 클라이언트 매칭에 관한 연구에서는 대부분 유사한 특성을 가진 클라이언트들을 클러스터링해 같은 그룹으로 만든다. 여기서 특성은 플레이 수준, 선호하는 행위, 플레이 패턴 등이 된다[16]. [17]에서는 매칭 기준이 주어지면, 클라이언트들을 그 기준에 부합한 정도에 따라 정렬한 후, 앞에서부터 차례로 그룹으로 묶는 단순한 방법을 제안했다. [15]에서는 클라이언트의 특성을 목표, 도전, 보상으로 구성된 구조체들의 트리 형태로 표현한 후, 유사성을 갖는 클라이언트들을 클러스터링하는 알고리즘을 제안했다. 유사한 특성을 가진 클라이언트들을 묶기 위해서는 기존 클러스터링 테크닉이 사용되는데, [18]에서는 K-means 클러스터링을 사용했다.

앞선 기존 연구의 결과와 본 논문에서 고려중인 무작위 매칭은 같이 사용할 수 있다. 즉, 신규 클라이언트의 수가 많은 게임의 경우, 먼저 클라이언트들을 특성에 따라서 여러 카테고리들로 분류한 후, 각 카테고리별로 본 논문에서 제안한 방법을 적용하면 될 것이다.

2. 문제 모델링

시간 인터벌 T 동안 새로 도착한 게임 클라이언트의 수는 N이고, 그들의 리스트를 $P = <p_1, p_2, \ldots, p_N>$ 라 한다. 게임 서버의 개수는 M개이고, 그들의 리스트는 $S=<s_1, \ldots, s_M>$라 한다. 서버 s_j와 클라이언트 p_i간의 거리를 $d(p_i,s_j)$로 표현한다.

상수 U는 그룹 플레이에서 그룹의 크기를 나타낸다. N은 U의 배수라고 가정한다. 총 클라이언트 그룹 수는 N/U가 되고, 각 그룹은 g_k ($1 \leq k \leq N/U$) 로 표현한다.

Find $x_{i,k}$'s and $y_{k,j}$'s, $1 \le a \le N$, $1 \le k \le N/U$, $1 \le j \le M$ such that

Minimize $DV = \sum_{a=1}^{N} \sum_{b=a+1}^{N} \frac{d_{a,b}}{U*(U-1)/2*N/U}$

Subject to

(*C1*) $U*|\sum_{k=1}^{N/U} y_{k,v} - \sum_{k=1}^{N/U} y_{k,w}| \le H * U$, $1 \le v \le M$, $1 \le w \le M$

(*C2*) $\sum_{k=1}^{N/U} x_{i,k} = 1$, $1 \le i \le N$

(*C3*) $\sum_{i=1}^{N} x_{i,k} = U$, $1 \le k \le N/U$

(*C4*) $\sum_{j=1}^{M} y_{k,j} = 1$, $1 \le k \le N/U$

(*C5*) $x_{a,k}=1 \wedge x_{b,k}=1 \wedge y_{k,j}=1 \Rightarrow d_{a,b}=|d(p_a,s_j) - d(p_b,s_j)|$, $1 \le a \le N$, $1 \le b \le N$, $1 \le k \le N/U$, $1 \le j \le M$

(*C6*) $d_{a,b} \ge 0$, $1 \le a \le N$, $1 \le b \le N$

(*C7*) $x_{i,k}=1$ or 0, $1 \le i \le N$, $1 \le k \le N/U$

(*C8*) $y_{k,j}=1$ or 0, $1 \le k \le N/U$, $1 \le j \le M$

[Fig. 1] Problem Modeling in IP Form

$x_{i,k}$는 클라이언트 p_i가 그룹 g_k에 하는 지를 나타내는 변수이다. $y_{k,j}$는 그룹 g_k가 서버 s_j에 할당되는 지를 나타내는 변수이다. 목적 함수(objective function)에 나오는 $d_{a,b}$는 두 클라이언트 p_a와 p_b가 통일한 서버에 배정되었을 때, 그들 간의 해당 서버와의 거리 편차이다. $d_{a,b}$는 조건 *C5*에 정의되어 있다. 따라서 *DV*는 같은 그룹에 속한 클라이언트들과 배정된 서버간의 평균 거리 편차를 나타낸다. 목적 함수는 *DV*를 최소화하는 것이다.

*C1*에 나오는 수식 $U*\sum_{k=1}^{N/U} y_{k,v}$와 $U*\sum_{k=1}^{N/U} y_{k,w}$는 서버 s_v와 s_w에 할당된 클라이언트의 수로서, 이들 서버의 부하를 나타낸다. 조건 *C1*은 서버들 간의 지나친 부하 불균형을 막기 위한 조건으로서, 상수 H는 서버들 간의 최대 부하 불균형 허용치이다. H=0으로 설정하면, 모든 서버는 동일한 부하를 갖도록 함을 의미한다. 조건 *C2*는 각 클라이언트 p_i는 그룹 한 개에만 속해야 함을, 조건 *C3*은 한 그룹은 *U*명의 클라이언트들로 구성됨을, *C4*는 각 그룹 g_k는 한 개 서버에만 속해야 함을 나타낸다.

3. GA 기반의 클라이언트-서버 배정 알고리즘

GA 기반 알고리즘에서는 먼저 초기 해 집단(population)을 형성한 후, 다음 세대의 해 집단을 구하는 과정을 반복한다. 과정 반복은 원하는 수준의 적합도(fitness)를 가진 해를 발견하거나 미리 정한 시간이나 회수까지만 계속된다. 각 해는 염색체(chromosome)로 표현되는데, 세부 표현 양식은 응용에 맞게 적절히 결정된다. 한 세대에서 다음 세대의 해 집단을 구하기 위해, 선택(selection), 교차(cross-over), 변이(mutation) 연산을 이용한다. 자세한 설명은 [4]에 기술되어 있다.

3.1 염색체 및 적합도의 정의

우리의 서버 배정 알고리즘에서 염색체는 2장에서 기술한 모든 $x_{i,k}$와 $y_{k,j}$ 들로 구성된 리스트 형태를 갖는다.

$<x_{1,1}, \ldots, x_{1,N/K}, x_{2,1}, \ldots, x_{N,N/K},$
$y_{1,1}, \ldots, y_{1,M}, y_{2,1}, \ldots, y_{N/K,M}>$

적합도는 2장에서 정의한 DV를 음수화한 $-DV$로 측정한다.

3.2 전체 절차

[Fig. 1]에 기술된 서버 배정 문제를 GA 기반으로 해결하는 방법은 [Fig. 2]의 *PlayerAlloc* 프

로시저와 같이 진행된다. 변수 *SOL*은 해 집단으로서 유전체의 집합을 저장한다. Step 1에서는 *PS*개의 유전체로 초기화 되는데, 초기 염색체들은 뒤에서 설명할 *FinishAlloc* 함수를 이용해서 생성된다. 이 함수는 거리 편차가 가장 작은 클라이언트 그룹과 해당 서버를 차례로 구해가는 휴리스틱 방법을 사용한다.

Step 2에서는 부모 세대에서 자식 세대 생성을 반복을 통해, *SOL*에 저장된 유전체의 적합도를 개선해 간다. 이 반복 작업은 최대 반복 회수인 *LOOP_THR*만큼 진행된다. Step 2-1에서는 *SOL*에 든 염색체들 중 적합도가 높은 절반을 선택해 변수 *NR*에 저장한다. *VF(·)* 함수는 주어진 유전체에 대한 적합도를 계산한다.

Step 2-2에서는 *NR*에 저장된 유전체를 대상으로 교차와 변이 연산을 각각 실행해 그 결과를 *NR2* 변수에 저장한다. 이들 연산은 *Cross_Over(·,·)* 및 *Mutation(·)* 함수로 구현된다. 이들 총 연산의 수는 *PS*/2이고, 교차 연산의 수는 변이 연산 수의 *CMR* 배이다. 각 세대에서 구하는 유전체의 총수인 *PS*, 연산의 비인 *CMR*, 반복 회수인 *LOOP_THR*은 *PlayAlloc* 프로시저에서 고려되는 되는 주요 파라미터들이다 이들의 값은 응용 상황에 맞게 적절히 설정된다. Step 2-3에서는 *NR*과 *NR2*를 합쳐 다음 세대의 유전체 집단을 완성한다.

Procedure *PlayerAlloc*
Input Parameters: *P, S, U, H, CMR, PS, LOOP_THR*

Step 1) set *SOL* to a set of *PS* initial chromosomes;
Step 2)
for *LOOP* = 1 **to** *LOOP_THR* {
Step 2-1) choose the *PS*/2 largest chromosomes *ch's* from *SOL* in terms of *VF*(*ch*), and let *NR* be a set of those chosen chromosomes;
Step 2-2) set *NR2* to an empty list;
for *l*=1 **to** (*PS*/2)**CMR*/(1+*CMR*) **do** {
choose two arbitrary elements ch_1 and ch_2 from *NR*;
ch_3 = *Cross_Over*(ch_1, ch_2);
add ch_3 to *NR2*;
}
for *l*=1 **to** (*PS*/2)/(1+*CMR*) **do** {
choose an arbitrary element ch_1 from *NR*;
ch_4 = *Mutation*(ch_1);
add ch_4 to *NR2*;
}
Step 2-3) *SOL* = *Concatenate*(*NR*, *NR2*);
}
Step 3) *CHR* = arg $\max_{ch \in SOL}$ *VF*(*ch*);
Step 4) output *CHR*

[Fig. 2] Procedure PlayerAlloc

3.3 연산에 대한 설명

우리의 클라이언트-서버 배정 알고리즘에서 선택 연산은 [Fig. 2]의 Step 2-1에 기술되어 있다. 교차 연산을 수행하는 *Cross_Over(·,·)* 함수는 [Fig. 3]에 기술되어 있는데, 이 함수는 두 개 염색체 c_1 및 c_2에 대해 교차 연산을 수행한 후 결과 염색체인 *ch*를 반환한다. Step 1에서는 *ch* 변수의 모든 요소들은 0으로 초기화하고, c_1과 c_2 중 적합도 가 큰 것을 *cl*, 작은 것을 *cs*로 정한다.

Step 2에서는 *cl*에서의 서버들을 *DV_SVR*(·,·) 결과에 따라 올림차순으로 정렬한다. *DV_SVR*(·,·)는 주어진 서버에 배정된 클라이언트들 간의 평균 거리 편차를 구하는 함수이다. *CNT*가 cs에서의 서버 s_j에 할당된 클라이언트들의 수라면, *DV_SVR*(*cs*, s_j)는 다음으로 구한다:

$$\sum_{each\ g_k \in s_j} \sum_{\substack{each\ pair\ of\ p_a\ and\ p_b \in g_k,\ a<b \\ and}} \frac{|d(p_a,s_j)-d(p_b,s_j)|}{CNT*(CNT-1)/2}$$

Step 3은 cl에서의 서버들 중 $DV_SVR(\cdot,\cdot)$가 적은 $M/2$개 서버들의 배정 상태를 ch에 그대로 적용한다. 서버의 배정 상태란 해당 서버에 어떤 그룹들이 배정되고, 그룹 구성원들은 누구인 지를 말한다. Step 4에서는 나머지 서버들에 대해, cs에서 이들에 배정된 클라이언트 그룹들 중 자격을 갖춘 그룹들의 배정 상태를 ch에 적용한다. 여기서 자격이란 모든 구성원들이 ch에 아직 배정되지 않았음을 말한다.

Step 5에서는 아직까지 서버 배정이 되지 않은 클라이언트들, 구성원이 확정된 않은 그룹들, Step 3에서 고려되지 않은 서버들을 대상으로, *FinishAlloc* 함수를 이용해 서버 및 그룹 배정을 마무리 한다.

Function *Cross_Over*(c_1, c_2)

Step 1)

let ch be an initial temporary chromosome where all $x_{i,k}$'s and $y_{k,j}$'s are 0;

e = 0;

if $VF(c_1) > VF(c_2)$ {cl = c_1; cs = c_2;} **else** {c_l = c_2; cs = c_1;}

Step 2) sort all the servers s_j's of cl in an increasing order of $DV\_SVR(cl, s_j)$, and let SL = $<s_{\pi(1)}, ..., s_{\pi(M)}>$ be the sorted list of the servers.

Step 3)

for j = 1 **to** $M/2$

for each group g_k allocated to $s_{\pi(j)}$ in c_l **do** {

e++;

$y_{e,\pi(j)}$ = 1 in ch;

for each player $p_i \in g_k$ in c_l **do** $x_{i,e}$ = 1 in ch;

}

Step 4)

for j = $M/2$ + 1 **to** M

for each group g_k allocated to $s_{\pi(j)}$ in cs such that $\sum_{each\ p_i \in g_k} \sum_{z=1}^{N/U} x_{i,z}$ = 0 in ch **do** {

e++;

$y_{e,\pi(j)}$ = 1 in ch;

for each player $p_i \in g_k$ in cs **do**

$x_{i,e}$ = 1 in ch;

Step 5) let TP = {p_i | p_i has not yet been allocated to a server in ch};

TS = {$s_{\pi(M/2+1)}$, ..., $s_{\pi(M)}$}

TG = {g_{e+1}, ..., $g_{N/U}$};

ch = *FinishAlloc*(ch, TP, TG, TS);

Step 6) return ch;

[Fig. 3] Function Cross_Over

[Fig. 4]는 변이 연산을 처리하는 Mutation 함수를 기술한다. Step 1에서는 염색체 ch에서 서버들 중 최대 거리 편차를 가지는 서버를 찾아 이를 s_j라 한다. Step 2에서는 서버 s_j의 클라이언트 배정을 무효화한 후, Step 3에서는 FinishAlloc 함수를 이용해, 이들 클라이언트들에게 새로운 매칭이 일어나도록 한다.

Function *Mutation*(ch)

Step 1) choose a server s_j with maximum $VF\_SVR(ch, s_j)$ from S in ch;

let TG = {g_k | g_k is a group allocated to s_j};

let TP = {p_i | p_i is a player in TG};

Step 2)

for each group g_k in TG **do** {

$y_{k,j}$ = 0 in ch;

for each player p_i belonging to g_k **do**

$x_{i,k}$ = 0 in *ch*;

}

Step 3) *ch* = *FinishAlloc*(*ch*, *TP*, *TG*, *S*);

Step 4) return *ch*;

[Fig. 4] Function Mutation

FinishAlloc 함수의 입력 파라미터는 4개로서, 이들을 ch, TP, TG 및 TS로 부르자. 여기서 TP은 염색체 ch에서 아직 그룹 및 서버가 정해지지 않은 클라이언트 집합, TG는 아직 멤버들이 정해지지 않은 그룹 집합, 그리고 TS는 TP에 속한 클라이언트들을 배정할 수 있는 서버들의 집합이다. FinishAlloc 함수의 역할은 TP의 클라이언트들에게 그룹과 서버를 배정하고, 완성된 ch를 반환하는 것이다. 자세한 동작은 다음 절차를 모든 클라이언트들이 서버에 배정될 때까지 반복한다.

Step 1: 새로운 클라이언트 그룹 한 개를 추가 배정하는데 적합한 서버들을 파악한다. 이런 서버는 클라이언트 배정 후 [Fig. 1]의 제약 조건 *C1*을 위반하지 않거나 또는 최대 서버 부하 편차를 증대시키지 않는 것이다.

Step 2: Step 1에서 파악한 각 서버에 대해, 아직 서버 배정이 되지 않은 클라이언트들만으로 구성되면서 평균 거리 편차가 가장 작은 그룹을 다음같이 찾는다. 클라이언트들을 해당 서버와의 거리 순으로 정렬한 후, 연이은 *U*명의 클라이언트들을 한 그룹으로 차례로 묶고 그들 간의 거리 편차를 서로 비교해 본다.

Step 3: Step 2에서 찾은 그룹들 중 가장 거리 편차가 작은 클라이언트 그룹을 해당 서버에 배정한다.

4. 실 험

우리는 [Fig. 2]에 기술된 클라이언트-서버 배정 방법의 특성과 유용성을 실험을 통해 분석했다. 실험에 사용한 파라미터는 게임 플레이를 원하는 신규 사용자 도착률(user arrival rate) λ, 서버 개수 M, 해 집단의 크기 PS, 교차 연산 수와 변이 연산 수의 비 CMR, 클라이언트 그룹의 크기 U, 서버 부하 불균형 허용치인 H이다. 사용자 도착률은 단위 시간 동안 발생하는 게임 플레이를 원하는 클라이언트들의 평균 명수를 나타낸다. 이들 파라미터에 다양한 값을 설정한 후, 평균 거리 편차인 DV를 비교해 보았다.

단위 시간 동안의 신규 클라이언트 수는 푸아송 분포(Poisson distribution)를 따른다고 가정한다. 편의상 클라이언트와 서버간의 거리는 1 - 10 사이의 값을 갖고, 평균값이 5인 균일 분포(Uniform distribution)를 갖는 것으로 가정했다. 우리의 방법은 특정 네트워크 토폴로지에 국한되지 않는다. 다중 서버 기반의 온라인 게임에서의 서버 클라이언트간의 실제 거리 분포 데이터를 획득할 수 없어 균일 분포를 사용했다. 서버들의 성능은 동일하고, 서버 성능은 최대 클라이언트 도착률을 처리할 정도로 충분히 크다고 가정한다. PlayAlloc 프로시저에서 LOOP_THR=50로 설정했다. 아래 그림들에 나오는 DV 값은 각 실험 세팅별로 20번을 반복 수행한 후 구한 평균이다.

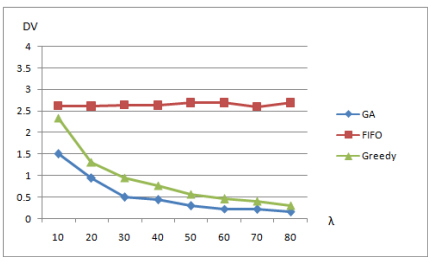
본 장에 기술된 모든 종류의 실험에는 [Fig. 4]의 방법뿐만이 아니라 다른 일반 방법 2개를 함께 시뮬레이션했다. 그 이유는 우리 방법의 성능을 이들과 비교해 얼마만큼의 개선이 있는 지를 파악해 보기 위함이다. 모든 그림에서 GA는 우리 방법을, FIFO와 Greedy는 비교 대상인 방법들을 나타낸다. FIFO는 도착순으로 클라이언트 U명씩을 한 그룹으로 묶은 후, 라운드-로빈 방식으로 서버를 선택해 이 그룹을 배정하는 방법을 말한다. Greedy는 FinishAlloc 함수에서와 같이 동작하는 휴리스틱 방법이다.

[Fig. 5]는 다양한 λ에 따라 구한 클라이언트-서버간의 거리 편차 및 기타 특성이다. 실험 파라미터는 M=10, PS=20, U=4, CMR=2, H=1로 설정했다. [Fig. 5] (a)에서 보다시피, GA는 Greedy나 FIFO 보다는 적은 거리 편차를 보였다.

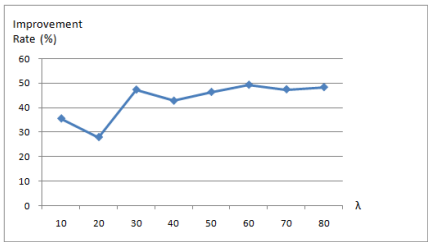
우리 방법과 Greedy에서는 λ가 증가하면 DV는 줄어든다. 이것은 λ가 크면 [Fig. 2]의 입력 파라미터인 P에 속하는 클라이언트 수가 늘어나고, 클라이언트 U명을 묶어 함께 서버 배정을 고려할 때에 거리 편차가 보다 적은 묶음을 발견할 기회가 높아지기 때문이다. FIFO에서는 λ의 증가가 DV에 미치는 영향이 거의 없다고 볼 수 있는데, 클라이언트들을 도착순으로 차례로 그룹화하기 때문일 것이다.

[Fig. 5] (b)는 GA의 DV의 Greedy의 DV에 대한 개선율을 보여준다. 이것은 $\frac{Greedy의\ DV - GA의\ DV}{Greedy의\ DV}$로 계산한 것이다. λ가 30까지는 개선율이 대체적으로 크게 증가하지만, 그 이상의 영역에서는 조금씩만 증가하는 경향을 보인다. 그 이유는 (a)에서 보디시피, λ가 30을 넘는 영역에서는 GA의 DV와 Greedy의 DV 사이의 차이가 서서히 줄어가기 때문이다.

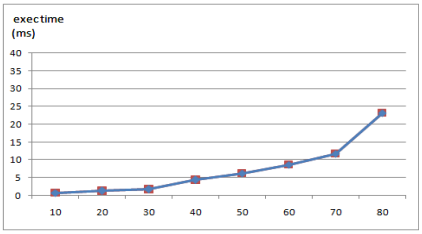
[Fig. 5] (c)는 시뮬레이션 하는데 걸린 시간을 보여준다. λ의 증가에 따라 거의 선형적으로 시간이 증가한다. 하지만 λ=80인 경우에도, 약 수십 ms 정도의 시간이면 서버 배정을 마칠 수 있었다. 만약 게임 플레이의 평균 시간이 30분, 단위 시간이 1초라면, λ=80은 동시 접속자 수가 144,000명 (80*30*60 = 144,000)인 경우를 의미한다.



(a) Client-Server Distance Deviation



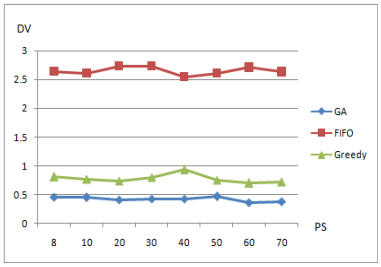
(b) Improvement Rate



(c) Execution Time

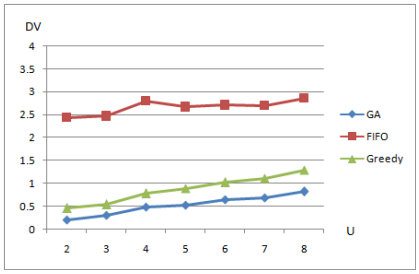
[Fig. 5] Client-Server Distance Deviation w.r.t Client Arrival Rate (M=10, PS=20, U=4, CMR=2, H=1)

[Fig. 6]은 해 집단의 크기인 *PS*가 거리 편차에 미치는 영향을 실험한 결과이다. 실험 파라미터는 앞선 실험과 같이 하면서, λ는 40으로 설정했다. 그림에서와 같이 *PS*가 증가가 하면 *DV*는 조금씩 줄어드는 경향을 보인다. 해 집단이 커지면 좋은 해를 포함할 확률이 높아지게 되고, 다음 세대의 해를 생성하기 위한 선택에서 좋은 해가 선택될 가능성도 높다. 하지만 줄어드는 경향이 현저하지 않는데, 그 이유는 [Fig. 2]에서 나타나듯이 우수한 유전체들이 우선적으로 선택되거나 생성되어 다음 세대에 포함되는 전략을 사용하기 때문이다. 즉 작은 해 집단이라도 최고 우수한 유전체는 큰 해 집단의 그것과 비슷한 수준의 *DV* 값을 가지고 있을 수 있다. 모든 *PS* 값에 대해서 *Greedy*와 *FIFO* 보다 나은 *DV*를 보이고 있다.



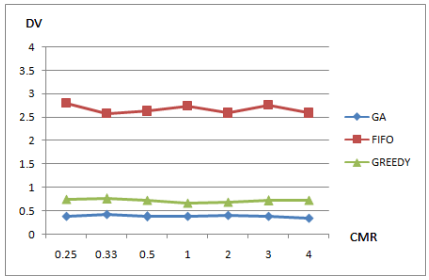
[Fig. 6] Client-Server Distance Deviation w.r.t Population Size (M=10, U=4, CMR=2, H=1, λ=40)

[Fig. 7]은 클라이언트 그룹의 크기인 *H*가 거리 차에 미치는 영향을 실험한 결과이다. *GA*나 *Greedy*의 경우, *H*가 증가가 하면 *DV*가 늘어나는 경향을 보인다. 이것은 한 서버에 대해 클라이언트들의 수가 커지면, 해당 서버와의 거리들이 그 만큼 다양해지고 이로 인해 계산되는 거리 편차가 커질 가능성이 있기 때문이다. *FIFO*의 경우, *H*가 증가가 하면 *DV*가 늘어나는 경향이 있지만 그 정도가 경미하다. *FIFO*는 도착순으로 클라이언트들을 그룹으로 묶기에 *DV* 값 자체가 커고, 이로 인해 앞에서 언급한 거리 편차가 증가하는 요인이 상대적으로 약화되기 때문일 것이다.



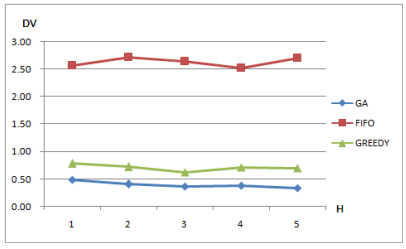
[Fig. 7] Client-Server Distance Variation w.r.t Group Size (M=10, PS=20, CMR=2, H=1, λ=40)

[Fig. 8]은 교차 연산 수와 변이 연산 수의 비인 *CMR*이 거리 편차에 미치는 영향을 보여준다. *FIFO*와 *Greedy*와는 같이, *GA*에서도 *CMR*의 증가에 따라서 *DV*가 감소하거나 증가하는 추세를 보여 주지 않는다. 따라서 우리의 방법에서는 교차 연산과 변이 연산이 해의 적합도 향상에 미치는 효과는 거의 동일하다고 볼 수 있다. 자연계에서는 교차가 변이보다는 많이 일어나므로, 본 장에서의 실험에서는 그런 값 중 하나인 2를 *CMR*로 설정했다.



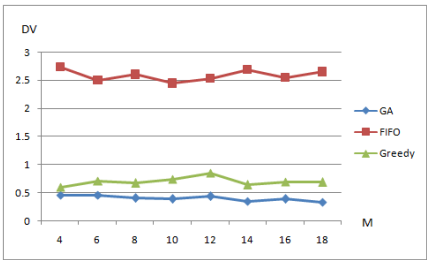
[Fig. 8] Client-Server Distance Deviation w.r.t Crossover Mutation Ratio (M=10, PS=20, U=4, H=1, λ=40)

[Fig. 9]는 서버 부하 불균형 허용치인 *H*가 거리 편차에 미치는 영향을 실험한 결과이다. *GA*의 경우, *H*가 증가하면 *DV*는 조금씩 줄어드는 경향을 보인다. *H*의 증가로 제약조건 *C1*을 만족하는 서버들이 늘어나기 때문에, 그 만큼 서버 배정을 위한 선택의 폭이 커져 *DV* 값의 개선을 기대할 수 있다. 실험에서는 서버들과 클라이언트들이 네트워크상에 고루 분산되어 있다고 가정한다. 따라서 부하 불균형을 심화시키면서 적용한 클라이언트-서버 배정에서도, 서버별로 클라이언트들간의 거리 편차는 현저히 개선되지 않는 것으로 보인다.



[Fig. 9] Client-Server Distance Deviation w.r.t Allowed Server Load Imbalance Factor (M=10, PS=20, U=2, CMR=2, λ=40)

[Fig. 10]은 총 서버의 개수인 M이 거리 편차에 미치는 영향을 보여준다. 실험 파라미터는 앞선 실험에서 같이 PS=20, U=2, CMR=2, H=1, λ=40로 설정했다. GA의 경우, M이 증가가 하면 DV는 조금씩 감소하는 경향을 보인다. 이것은 서버의 개수가 많아지면, 주어진 클라이언트 그룹에 대해 보다 거리 편차가 작은 서버를 찾을 있는 확률이 높아지기 때문이다. 앞에서 언급한대로 서버들과 클라이언트들의 네트워크상에 고루 분산되었다고 가정하기에, 서버의 수가 늘어도 서버별 클라이언트들 간의 거리 편차는 큰 폭으로 개선되지 않는 것으로 판단된다.



[Fig. 10] Client-Server Distance Deviation w.r.t Server No (PS=20, U=2, CMR=2, H=1, λ=40)

5. 결 론

많은 클라이언트들이 동시에 플레이하는 온라인 게임에서는, 보통 다중 서버 구조를 채택한다. 이 때 그룹을 이룬 후 그룹내 다른 사람들과 경쟁하는 방식의 경우, 플레이의 공정성 및 흥미를 위해서 클라이언트-서버간의 사용자 반응시간의 균등함이 중요하다. 사용자 반응시간은 클라이언트와 서버간의 거리와 서버 부하에 큰 영향을 받는다.

본 논문에서는 클라이언트 매칭을 지원하면서 클라이언트-서버간의 거리 편차를 최소화하는 클라이언트-서버 배정 문제를 IP로 모델링하고, 그 해를 구하는 GA 기반의 방법을 제안했다. 우리의 IP 모델에는 서버 부하 균등에 관한 제약조건을 포함함으로써, 클라이언트를 서버에 배정 시 서버 부하 균등도 함께 고려하였다.

우리의 방법을 다양한 파라미터 세팅하에서 실험한 결과, FIFO나 탐욕(greed)의 기반의 방법보다는 좋은 성능을 보였다. 클라이언트 도착률이 높은 경우에도 수십 ms 안에 해를 구할 수 있기에, 실시간 응용에서 사용하기에 지장이 없음을 알 수 있었다. 또한, 해 집단의 크기, 서버의 개수, 허용 가능한 부하 불균형 지수가 커지면, 클라이언트-서버간의 거리 편치는 줄어드는 경향을 보인다. 반면에 클라이언트 그룹의 크기가 커지면, 거리 편차는 늘어나는 경향을 보였다.

우리가 조사해 본 바, 클라이언트 매칭과 클라이언트-서버간 거리 편차를 부하 균등과 함께 고려한 서버 배정에 관한 기존 연구는 거의 없었다. 거리 편차 최소화로 통해 얻는 게임 플레이의 성능 개선에 대한 분석은 추후 연구에서 다룬다.

ACKNOWLEDGMENTS

This work was supported by Hankuk University of Foreign Studies Research Fund of 2015.

REFERENCES

[1] S. Farlow, J. L. Trahan, "Client-Server Assignment in Massively Multiplayer Online Games", Proc. of CGAMES, 2014, pp.1-8.

[2] C. Eduardo B. Bezerra, et. al, "Adaptive Load-balancing for MMOG Servers Using KD-trees", ACM Computers in Entertainment, Vol. 10, No. 3, Article 5, 2012, pp. 1-16.

[3] R. W.H. Lau, "Hybrid Load Balancing for Online Games", Proc. of ACM International Conference on Multimedia, 2010, pp.1231-1234.

[4] Genetic Algorithm, https://en.wikipedia.org/wiki/Genetic_algorithm.

[5] G. Armitage. "Optimising Online FPS Game Server Discovery through Clustering Servers by Origin Autonomous System." Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), 2008, pp. 3-8.
[6] S. Gargolinski, C. St. Pierre, M. Claypool. "Game Server Selection for Multiple Players." Proc. of 4th ACM SIGCOMM Workshop on Network and System Support for Games, 2005, pp. 1-6.
[7] Y. Chen, S. Radhakrishnan, S. Dhall, S. Karabuk. "Server Selection with Delay Constraints for Online Games." Proceedings of GLOBECOM Workshops, 2010, pp. 882-887.
[8] M. Kohana, et. al, "Dynamic ReAssignment Rules on Multi-Server Web-based MORPG System", International Journal of Grid and Utility Computing, vol. 3, no. 2/3, pp. 136-144, 2012
[9] L. Zhang, X. Tang, "The Client Assignment Problem for Continuous Distributed Interactive Applications: Analysis, Algorithms, and Evaluation," IEEE Trans. Par. Distrib. Sys., vol. 25, no. 3, 2014, pp.785-795.
[10] K. Lee, B. Ko, S. Calo. "Adaptive Server Selection for Large Scale Interactive Online Games." Computer Networks, Vol. 49, Issue 1, 2005, pp.84-102.
[11] D. T. Ahmed "A microcell oriented load balancing model for collaborative virtual environments", Proc. of VECIMS. 208, pp.86-91.
[12] B. De Vleeschauwer, et. al, "Dynamic Microcell Assignment for Massively Multiplayer Online Gaming", Proc. of NetGames, 2005, pp. 1-7.
[13] B. Ng, A. Si, R. Lau, F. Li, "A Multi-Server Architecture for Distributed Virtual Walkthrough", Proc. of the ACM Symposium on Virtual Reality Software and Technology, 2002, pp.11-13
[14] D. Lee, M. Lim, S. Han, K. Lee, "ATLAS: A Scalable Network Framework for Distributed Virtual Environments", Presence: 16(2), 2007, pp.125-156
[15] S.-Y. Yun, A Proutiere, "Distributed Proportional Fair Load Balancing in Heterogenous Systems", Proc. of SIGMETRICS, 2015, pp.17-30.
[16] Y Francillette, "A Players Clustering Method to Enhance the Players' Experience in Multi-player Games", Proc. of CGAMES, 2013, pp.229-234.
[17] C. Schlup. "Automatic Game Matching", http://dcg.ethz.ch/theses/ws0203/OnlineMatching_abstract.pdf.
[18] R. S. Anders Drachen, "Guns, Swords and Data: Clustering of Player Behavior in Computer Games in the Wild," Proc. of IEEE Conference on Computational Intelligence and Games (CIG), 2012, pp. 163-170.

이 성 해(Sunghae, Lee)

2014- 한국외국대학교 컴퓨터공학전공 석사과정
2014- 라움 대표

관심분야 : 기능성 게임, 웹 및 앱 정보 시스템

김 상 철(Kim, Sangchul)

1994 미시간주립대학교 컴퓨터공학과 박사
1983-1994 ETRI 연구원
1994- 한국외국대학교 컴퓨터공학과 교수

관심분야 : 기능성게임, 게임 AI, 멀티미디어시스템