Dynamic Multi-Service Load Balancing in Cloud-based Multimedia System

Chun-Cheng Lin, *Member, IEEE*, Hui-Hsin Chin, *Student Member, IEEE*, Der-Jiunn Deng, *Member, IEEE*

***Abstract*—Consider a centralized hierarchical cloud-based multimedia system (CMS) consisting of a resource manager, cluster heads, and server clusters, in which the resource manager assigns clients' requests for multimedia service tasks to server clusters according to the task characteristics, and then each cluster head distributes the assigned task to the servers within its server cluster. For such a complicated CMS, however, it is a research challenge to design an effective load balancing algorithm which spreads the multimedia service task load on servers with the minimal cost for transmitting multimedia data between server clusters and clients, while the maximal load limit of each server cluster is not violated. Different from previous works, this paper takes into account a more practical dynamic multi-service scenario in which each server cluster only handles a specific type of multimedia tasks, and each client requests a different type of multimedia services at different time. Such a scenario can be modelled as an integer linear programming problem, which is computationally intractable in general. As a consequence, this paper further solves the problem by an efficient genetic algorithm with immigrant scheme, which has been shown to be suitable for dynamic problems. Simulation results demonstrate that the proposed genetic algorithm can efficiently cope with dynamic multi-service load balancing in CMS.**

***Index Terms*—Genetic algorithm, metaheuristic, load balancing, cloud computing, multimedia system.**

I. Introduction

With advance of technology, cloud-based multimedia system (CMS) [1] emerges because of a huge number of users' demands for various multimedia computing and storage services through the Internet at the same time, e.g., see [2], [3], [4]. It generally incorporates infrastructure, platforms, and software to support a huge number of clients simultaneously to store and process their multimedia application data in a distributed manner and meet different multimedia QoS requirements through the Internet. Most multimedia applications (e.g., aduio/video streaming services, etc.) require considerable computation, and are often performed on mobile devices with constrained power, so that the assistance of cloud computing is strongly required. In general, cloud service providers offer the utilities based on cloud facilities to clients, so that clients do not need to take much cost to request multimedia services and process multimedia data as well as their computation results. By doing so, multimedia applications are processed on powerful cloud servers, and the clients only need to pay for the utilized resources by the time.

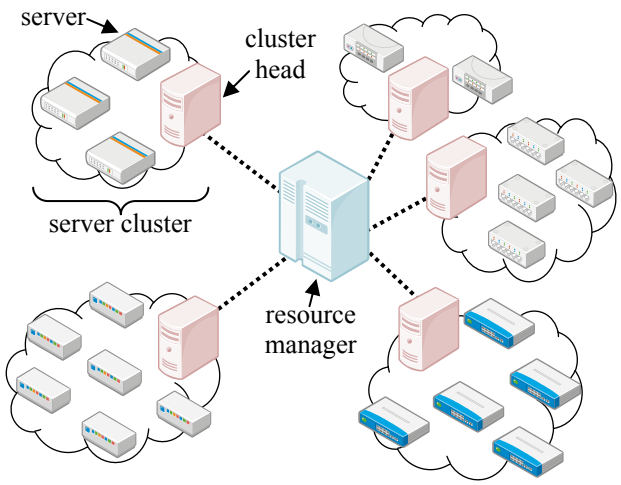

Fig. 1. Illustration of a centralized hierarchical cloud-based multimedia system.

This paper considers a centralized hierarchical CMS (as shown in Figure 1) composed of a *resource manager* and a number of *server clusters*, each of which is coordinated by a *cluster head*, and we assume the servers in different server clusters to provide different services. Such a CMS is operated as follows. Every time when the CMS receives clients' requests for multimedia service tasks, the *resource manager* of the CMS assigns those task requests to different *server clusters* according to the characteristics of the requested tasks. Subsequently, the *cluster head* of each *server cluster* distributes the assigned task to some server within the server cluster. It is not hard to observe that the load of each server cluster significantly affects the performance of the whole CMS. In general, the resource manager of the CMS is in pursuit of fairly distributing the task load across server clusters, and hence, it is of importance and interest to be able to cope with load balancing in the CMS.

Load balancing for wireless networks has been studied extensively in the previous literature, e.g., multiple-factor load balancing [5], load balancing with policy mechanism [6], load balancing based on game theory [7], load balancing in WLANs [8], multi-service load balancing [9] and soft load balancing [10], and scheduling [11] in heterogeneous wireless networks,

Manuscript received August 1, 2012; revised December 20, 2012; accepted March 3, 2013.

C.-C. Lin is with the Department of Industrial Engineering and Management, National Chiao Tung University, Hsinchu 300, Taiwan. E-mail: cclin321@nctu.edu.tw

H.-H. Chin and D.-J. Deng are with the Department of Computer Science and Information Engineering, National Changhua University of Education, Changhua 500, Taiwan. D.-J. Deng is the corresponding author of this paper. E-mail: djdeng@cc.ncue.edu.tw

among others. Some previous works have also existed on load balancing for CMSs, e.g., see [3], [12]. Among them, the load balancing problem for CMSs in [3] is concerned with spreading the multimedia service task load on servers with the minimal cost for transmitting multimedia data between server clusters and clients, while the maximal load limit of each server cluster is not violated. A simplified concern in their setting is to assume that all the multimedia service tasks are of the same type. In practice, however, the CMS offers services of generating, editing, processing, and searching a variety of multimedia data, e.g., hypertext, images, video, audio, graphics, and so on [1]. Different multimedia services have various requirements for the functions provided by the CMS (storage, central processing unit, and graphics processing unit clusters), e.g, the QoS requirement of hypertext webpage services is looser than that of video streaming services. In addition, the settings in the previous works [3], [12] did not consider that load balancing should adapt to the time change.

To respond to the practical requirements mentioned above, we assume that in the CMS, each server cluster can only handle a specific type of multimedia service tasks, and each client requests a different type of multimedia services at different time. At each specific time step, such a problem can be modelled as an integer linear programming formulation, which is computationally intractable in general [13]. Conventionally, intractable problems are usually solved by meta-heuristic approaches, e.g., simulated annealing [14], genetic algorithm [15], particle swarm optimization [16], [17], etc. In this paper, we propose a genetic algorithm (GA) for the concerned dynamic load balancing problem for CMSs. GA has already found applications in a variety of areas in computer science and engineering, such as fast covariance matching [18], aircraft ground service scheduling problem [19], optimal electric network design [20], among others. In our setting of GA, elite immigrants and random immigrants are added to new population, because they are suitable for solving the problems in dynamic environments, e.g., see [21]. The experimental results show that to a certain extent, our approach is capable of dynamically spreading the multimedia task load evenly.

Note that some previous works on other issues of cloud computing or distributed computing have also existed, e.g., cost-optimal scheduling on clouds [22], load balancing for distributed multiagent computing [23], communication-aware load balancing for parallel applications on clusters [24], among others. Also note that GA has been applied to dynamic load balancing in [25], but their GA was designed for distributed systems, not specific to the CMS. In addition, they did have any multi-service concern.

II. Problem Description

Our load balancing problem for the CMS is based upon [3], which, however, only considered that all the multimedia service tasks are of the same type, and did not consider the dynamic scenario where load balancing should adapt to the time change. By extending their model with these concerns, this section first gives the system overview of the CMS, and then formulates our concerned problem.

A. System Overview

In general, CMSs can be divided into two categories: *centralized* and *decentralized*. This paper considers a centralized CMS as illustrated in Figure 1, which consists of a *resource manager* and a number of *server clusters* each of which is coordinated by a *cluster head*. Different from the decentralized CMS, every time when receiving clients' requests for multimedia service tasks, the *resource manager* of the centralized CMS stores the global service task load information collected from *server clusters*, and decides the amount of client's requests assigned to each server cluster so that the load of each server cluster is distributed as balanced as possible, in term of the cost of transmitting multimedia data between server clusters and clients. The decision of assignment is based upon the characteristics of different service requests and the information collected from server clusters.

In comparison to decentralized framework, the centralized framework is scalable as fewer overheads are imposed on the system, and hence, a lot of applications have existed, e.g., see [26]. However, the centralized framework has lower reliability since the load balancing algorithms may be dysfunctional due to the failure of the resource manager [25]. Although a decentralized framework is suitable to smaller systems, it is still easier to implement.

B. Problem Formulation

To formulate the CMS that can adapt to time dynamics, we assume time to be divided into different time steps. At the t-th time step, the CMS can be modelled as a complete weighted bipartite graph $G_t = (U, V, E, \phi, \psi^t, q, r^t, w^t)$ in which

- U is the set of vertices that represent the server clusters of the CMS;
- V is the set of vertices that represent clients;
- E is the set of edges between U and V, in which each edge $e_{ij} \in E$ represents the link between server cluster $i \in U$ and client $j \in V$;
- $\phi : U \to \mathbb{N}$ is a function used to restrict that server cluster i can only cope with multimedia tasks of type ϕ_i;
- $\psi^t : V \to \mathbb{N}$ is a function used to represent that client j requests the multimedia service of type ψ_j^t at the t-th time step;
- $q : U \cup V \to \mathbb{N}$ is a function used to represent that server cluster i can provide the multimedia service of QoS q_i;
- $r^t : U \cup V \to \mathbb{N}$ is a function used to represent that client j requests the multimedia service of QoS requirement r_j^t at the t-th time step;
- $w^t : E \to \mathbb{R}^+$ is the weight function associated with edges, in which w_{ij}^t denotes the w^t value that represents the cost for transmitting multimedia data between server cluster i and client j at the t-th time step, which is defined as follows:

$$w_{ij}^t = \begin{cases} \infty, & \text{if } d_{ij}^t \to \infty \text{ or } \phi_i \neq \psi_j; \\ d_{ij}^t l_{ij}^t, & \text{otherwise.} \end{cases} \quad (1)$$

where d_{ij}^t is the network proximity between server cluster i and client j; l_{ij}^t is the traffic load of the link between

server cluster i and client j that is defined as follows:

$$l_{ij}^t = \sum_{k \in K_i} u_{ikj}^t C_{ik} \quad (2)$$

where K_i is the set of servers in server cluster i; u_{ikj}^t is the server utilization ratio of server k in server cluster i due to client j, and C_{ik} is its capacity.

Note that the proximity d_{ij}^t between server cluster i and client j in Equation (1) is required to be measured at every time step due to dynamic change of network topology. This paper continues applying the setting of [3] based upon the *distributed binning scheme* in [27] to calculate the proximity d_{ij}^t. Like other previous works, we measure the proximity between the server cluster and the client as a distance between them. Take an example to explain how to calculate the proximity as follows. Here, we say that a node may be a server cluster or a client. First, we measure the distance of a node to a given set of landmark nodes in the network by the network link latency. Suppose that there are three landmarks in the network. The latencies from the concerned node to the three landmarks are 45, 10, and 25, respectively. Nodes are ranked according to the latency information: range 0 for latencies in [0, 15], range 1 for latencies in (15, 40], and range 2 for latencies higher than 40. Hence, the landmark order of the concerned node is "201". By using the landmark order, all the nodes can be classified into different bins, i.e., the nodes with the same landmark order fall into the same bin. By doing so, we only calculate the proximity between two nodes in the same bin, while the others in different bins mean that they are too far to communicate with each other, so their proximity is infinity.

With the above notations, the mathematical model of our concerned problem at the t-th time step can be stated as the following integer linear programming formulation:

$$\begin{aligned}
\text{Minimize} \quad & \lambda \frac{\sum_{i \in U} \sum_{j \in V} x_{ij}^t w_{ij}^t}{\sum_{j \in V} w_{\max}} \\
& + (1-\lambda)\left(1 - \frac{\sum_{j \in V} \sum_{i \in U} x_{ij}^t}{|V|}\right) \quad (3) \\
\text{subject to} \quad & \sum_{i \in U} x_{ij}^t \leq 1, \forall j \in V, \quad (4) \\
& \sum_{j \in V} x_{ij}^t l_{ij}^t \leq \sum_{k \in K_i} C_{ik}, \forall i \in U \quad (5) \\
& x_{ij}^t \phi_i = x_{ij}^t \psi_j^t, \forall i \in U, j \in V \quad (6) \\
& x_{ij}^t q_i \geq x_{ij}^t r_j^t, \forall i \in U, j \in V \quad (7) \\
& x_{ij}^t \in \{0,1\}, \forall i \in U, j \in V \quad (8)
\end{aligned}$$

where x_{ij}^t is an indicator variable defined as follows:

$$x_{ij}^t = \begin{cases} 1, & \text{if client } j \text{ is assigned to server cluster } i \\ & \text{at the } t\text{-th time step;} \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

In the above model, indicator variable x_{ij}^t (see Equation (8)) is used to determine whether to assign the link e_{ij} between server cluster i and client j in the complete bipartite graph $U \times V$. The objective (3) of the model is a weighted sum of two terms: the first is to minimize the total weighted values of the bipartite graph, i.e., to minimize the total cost of transmitting multimedia data at the t-th time step, while the second is to maximize the number of link assignments. Note that we let $w_{\max}$ be the maximal possible weight (less than infinity), and hence, the denominators of the two terms of the objective are used for normalizing them to the range $[0, 1]$, and $\lambda \in [0, 1]$ is used to adjust the weights of the two terms, so that the objective value always falls into the range $[0, 1]$. Constraint (4) guarantees that each client only allows at most one link to be assigned. For each client j in V, the constraint enforces that x_{ij}^t of at most one server cluster i is 1. Constraint (5) enforces that the utilized capacity of each server cluster cannot exceed its capacity at the t-th time step. Constraint (6) enforces that the multimedia service type requested by each client j is consistent with that provided by server cluster i. Constraint (7) enforces that each client j requests the multimedia server of the QoS no more than that offered by server cluster i.

As our model is rooted from the work in [3], the differences of our model from theirs are explained as follows.

- Different from the work in [3], we additionally consider four functions ϕ, ψ^t, q and r^t.
- About the link assignment to each client, the model in [3] constrains each client to be assigned to exactly one link, while ours allows each client to be assigned to one or zero link (Constraint (4)). That is, the previous model guarantees to serve each client, but ours does not, because our concerned problem is more complicated.
- With the above constraint, our objective additionally considers to maximize the number of link assignments, i.e., the number of served clients (see the second term in Objective (3)).
- Our model additionally considers Constraints (6) and (7) for multiple service types and QoS requirements, respectively.
- The load balancing algorithm in [3] is not adaptive, but ours is robust with time change, as the time change can be seen via the superscript t in the model.
- We allow mobility of clients, i.e., clients can change their locations at different time steps. Note that the problems that consider mobility of nodes have received much attention recently, e.g., see the survey in [28].

As a result, our concerned problem can be stated as follows:

DYNAMIC MULTI-SERVICE LOAD BALANCING IN CMS (CMS-DYNMLB): *Given a CMS with m server clusters and n clients, for $t = 1, 2, \cdots$, the bipartite graph $G_t = (U, V, E, \phi, \psi^t, q, r^t, w^t)$ underlies the CMS at the t-th time step (as described above) in which clients have mobility, while the link between clients and server clusters need be assigned. The objective of the problem is to assign multimedia service load so that total cost of transmitting multimedia data is minimized and the number of served clients is maximized.*

Since the CMS-dynMLB problem at each fixed time step can be modelled as an integer linear programming problem as mentioned above, it is computationally intractable in general [13], i.e., there does not exist any efficient deterministic

polynomial time algorithm for the problem. Hence, this paper proposes a genetic algorithm (GA) [15] with immigrant scheme [21] for solving the problem. The GA is a stochastic global search method that has proved to be successful for many kinds of optimization problems. GA is categorized as a global search heuristic. It works with a population of candidate solutions and tries to optimize the answer by using three basic principles, including selection, crossover, and mutation. For more details on GA, readers are referred to [15].

III. Our Genetic Algorithm with Immigrant Scheme

This section first gives our proposed algorithm, and then gives basic definitions and main components of the algorithm.

A. Our Algorithm

The basic idea of the GA is to imitate the evolutional behavior of a population of chromosomes (each of which represents a candidate solution) to find the solution close to the global optimal solution by using three basic evolutional operations: *selection*, *crossover*, and *mutation*. The key principle of the GA is that the fittest chromosomes can survive to the next generation, and in general the fittest chromosome in the final generation represents the final solution that has a good performance. We first define how a *chromosome* represents a candidate solution, and how the *fitness* of each chromosome translates the objective value of the corresponding solution. The initial step of the algorithm is to maintain a population (generation) of chromosomes that are initialized randomly or in some way. Next, a number of the chromosomes in the population are *selected* as the parental pool, in which each pair of chromosomes are *crossovered* to produce child chromosomes. Next, parts of the original chromosomes and the child chromosomes constitute the next generation. After evolving a maximal number of generations or achieving a convergent condition, the solution represented by the chromosome with the best fitness value in the last generation is outputted as the final solution.

Based on the above idea for GA, our dynamic load balancing algorithm is given in Algorithm 1, which is explained as follows. At each time step t, Algorithm 1 iterates on t to reallocate the network load assignments so as to adapt to the time change. At first, Line 2 constructs a complete weighted bipartite graph G_t, described in the previous section. Subsequently, we remove infeasible cases and then apply our GA to finding a locally optimal load assignment solution. Note that feasible solutions are restricted to Constraints (4), (5), (6), and (7). In Algorithm 1, Line 3 removes the links in G_t violating Constraints (6) and (7), while the other two constraints will be considered in our GA (Algorithm 3). Before using our GA to calculate solutions, the information of $\{l_{ij}^t\}$ and $\{w_{ij}^t\}$ is required, so Line 4 of Algorithm 1 calls Algorithm 2 to obtain those information. After that, Line 5 of Algorithm 1 calls the algorithm detailed in Algorithm 3 to compute our final load assignment solution $\{x_{ij}^t\}$.

In what follows, we explain how Algorithm 2 computes $\{l_{ij}^t\}$ and $\{w_{ij}^t\}$. At first, Line 1 considers each client $j \in V$

Algorithm 1 Dynamic Load Balancing Algorithm

1: **for** $t = 1, 2, \cdots$ **do**
2: consider complete weighted bipartite graph G_t
3: remove the links in G_t violating Constraints (6) and (7)
4: calculate $\{l_{ij}^t\}$ and $\{w_{ij}^t\}$ by calling Algorithm 2
5: assign $\{x_{ij}^t\}$ by calling Algorithm 3
6: **end for**

Algorithm 2 Calculate Weights

1: **for** each client $j \in V$ **do**
2: measure the latency from client j to each landmark
3: compute the landmark order ℓ_j of client j
4: obtain the set of available server clusters U_j
5: **for** each $i \in U_j$ **do**
6: measure the latency from server cluster i to each landmark
7: compute the landmark order ℓ_i of server cluster i
8: **if** $\ell_i = \ell_j$ **then**
9: measure the network proximity d_{ij}^t between server cluster i and client j
10: measure server utilization ratios u_{ikj}^t for all $k \in K_i$
11: calculate l_{ij}^t and w_{ij}^t by Equations (2) and (1), respectively
12: **else**
13: $w_{ij}^t = l_{ij}^t = \infty$
14: **end if**
15: **end for**
16: **end for**

to compute its weight w_{ij}^t with each server cluster i. Remind that we apply the distributed binning scheme to calculating the proximity, detailed in the previous section. Hence, Lines 2–3 calculate the landmark order ℓ_j of client j, and then, for each available server cluster i in the set U_j that include the server clusters connected to client j, Lines 6–7 calculate the landmark distance ℓ_i of server cluster i. In Lines 8–14, if $\ell_j = \ell_i$ (i.e., client j and server cluster i belong to the same landmark bin), we actually measure the network proximity between them, and then compute their l_{ij}^t and w_{ij}^t values; otherwise, we directly let the l_{ij}^t and w_{ij}^t values be ∞.

With the values of $\{l_{ij}^t\}$ and $\{w_{ij}^t\}$ computed by Algorithm 2, we are ready to apply the GA to computing the optimal load assignment in Algorithm 3, which is explained as follows. The GA runs differently based on two parameters: the input bipartite graph G_t and the input time step t. Remind that G_t may not be a complete graph any longer, because the links that violates some problem constraints have been removed in Line 3 of Algorithm 1. Let η and τ denote the size of a population and the number of generations, respectively. In Lines 1–5, the initial population of η chromosomes are generated in two cases. If $t = 1$ (i.e., this is the first time to run the GA), then we randomly generate the initial population which satisfies the remaining two constraints that we did not consider yet (i.e., Constraints (4) and (5)). Otherwise (i.e., $t \neq 1$, which implies that there existed the final population $\mathcal{P}_\tau^{t-1}$ of the $(t-1)$-th time step), Line 4 uses $\mathcal{P}_\tau^{t-1}$ as the

Algorithm 3 GENETIC ALGORITHM (GRAPH G_t, TIME STEP t)

1: **if** $t = 1$ **then**
2: generate and evaluate the initial population $\mathcal{P}_0^t$ of size η in which each chromosome has to satisfy Constraints (4) and (5).
3: **else**
4: $\mathcal{P}_0^t \leftarrow \mathcal{P}_\tau^{t-1}$
5: **end if**
6: $i \leftarrow 0$
7: **while** $i < \tau$ or the convergent condition is not achieved **do**
8: *select* the parental pool $\mathcal{Q}_i^t$ from $\mathcal{P}_i^t$
9: reproduce a new population $\mathcal{P}_i^t$ of size η by performing *crossover* procedure on pairs of chromosomes in $\mathcal{Q}_i^t$ with probability p_c
10: perform *mutation* procedure on chromosome in $\mathcal{P}_i^t$ with probability p_m
11: *repair* each infeasible chromosome in $\mathcal{P}_i^t$
12: evaluate $\mathcal{P}_i^t$
13: **if** $r_e > 0$ **then**
14: a number of the best chromosome E_{i-1}^t, called *elite*, are selected from the previous generation P_{i-1}^t
15: generate and evaluate $r_e \cdot \eta$ elite immigrants
16: **end if**
17: **if** $r_r > 0$ **then**
18: generate and evaluate $r_r \cdot \eta$ random immigrants
19: **end if**
20: replace the worst chromosomes in $\mathcal{P}_i^t$ with the above elite and random immigrants
21: $\mathcal{P}_{i+1}^t \leftarrow \mathcal{P}_i^t$
22: $i \leftarrow i + 1$
23: **end while**
24: output the best found chromosome as the solution at the t-th time step

initial population at the t-th time step.

Subsequently, the while loop in Lines 7–23 repeats at most τ iterations, each of which produces a population $\mathcal{P}_{i+1}^t$ of chromosomes, i.e., the next population at the t-th time step. Line 8 *selects* a number of chromosomes from the population $\mathcal{P}_i^t$ as the parental pool $\mathcal{Q}_i^t$. Lines 9 and 10 perform *crossover* and *mutation* operators to the population $\mathcal{P}_i^t$ with probabilities p_c and p_m, respectively. The *selection*, *crossover*, and *mutation* are conventional evolutionary operators of the GA. The details of *selection*, *crossover*, and *mutation* will be explained in the following subsections. In addition, since our concerned problem considers dynamic scenarios at different time steps, we add elite immigrants and random immigrants to adapt to dynamic changes, as the immigrants are used to solve dynamic problems conventionally [21]. Lines 13–16 add elite immigrants for increasing efficiency of convergence, while Lines 17–19 add random immigrants for increasing the population diversity. Line 20 replaces the worst chromosomes in $\mathcal{P}_i^t$ with the elite and random immigrants. After finishing the while loop, the best found chromosome is outputted as the solution at the t-th time step.

Note that a multimedia service task may not be able to be finished within a single time step, i.e., it takes a number of time steps to be finished. Hence, it is not necessary to maintain a specific link for the same service for some time steps because the network is packet-based. This flexibility is beneficial in the sense that rearrangement of cluster service load might result in achievement of smaller cost of transmitting data.

B. Basic Definitions of our GA

To use GA to solve the CMS-dynMLB problem, we first define the basic elements of GA (i.e., population, chromosome, fitness function) for the problem as follows.

1) Population: One population represents one generation. A population consists of a number of chromosomes, and the number of chromosomes depends on the given initial population size. In this paper, we use η to denote the number of chromosomes.

2) Chromosome: A solution for the CMS-dynMLB problem consists of all the indicator variables $\{x_{ij}^t | \forall i \in U, j \in V\}$. The solution is represented by a chromosome in GA. Recalling that $|U| = m$ and $|V| = n$, the solution for indicator variables $\{x_{ij}^t\}$ is encoded as a sequence of decimal numbers of length n: $\langle \sigma_1, \cdots, \sigma_j, \cdots, \sigma_n \rangle$ where $\sigma_j \in \{0, 1, 2, \cdots, m\}$ represents the link assignment of client j with the following two cases:

- if $\sigma_j = 0$, then each $x_{ij}^t = 0$ for any $i \in U$, i.e., client j is not linked at the t-th time step;
- otherwise, $\sigma_j = k \neq 0$, meaning that $x_{kj}^t = 1$ and $x_{ij}^t = 0$ for any $i \neq k$, i.e., client j is linked to server cluster k uniquely.

That is, all the x_{ij}^t values for a fixed j can be determined based on the value of σ_j. By doing so, it is guaranteed that $\sum_{i \in U} x_{ij}^t \leq 1, \forall j \in V$ (Constraint (4)).

For making the represented solution to be feasible, we require the chromosome to satisfy all the problem constraints. Since the cases violating Constraints (6) and (7) have been removed in Line 3 in Algorithm 1, the only remaining concern to guarantee the solution feasibility of the chromosome is to satisfy Constraint (5). Since computation of Constraint (5) requires the information of $\{l_{ij}^t\}$, the constraint is checked in Algorithm 3, after calling Algorithm 2 to calculate $\{l_{ij}^t\}$. For those chromosomes violating Constraint (5), we apply the *repairing* procedure to revising them to be feasible in Line 11 of Algorithm 3. The *repairing* procedure will be detailed in the next subsection.

3) Fitness Function: Fitness function is the measure for determining which chromosomes are better or worse. This function could control the trend of population development, so it is an important part of GA. It should be noted that the 'fitness' value in our definition is abused to be defined as the penalty for the bad quality of chromosome. That is, a larger 'fitness' value implies a worse chromosome. Hence, our GA aims to find the chromosome with minimal 'fitness' value. We let the objective (3) of our concerned problem as our fitness function as follows: $f(X(t)) = \lambda \cdot \sum_{i \in U} \sum_{j \in V} x_{ij}^t w_{ij}^t / \sum_{j \in V} w_{\max} + (1 - \lambda) \cdot (1 - \sum_{j \in V} \sum_{i \in U} x_{ij}^t / |V|)$ where $X(t)$ is the profile of $\{x_{ij}^t\}$.

C. Main Components of Our GA

The main components of our GA is introduced as follows.

1) Initialization: In Lines 1–5 of Algorithm 3, there are two cases for producing the initial population at each time step. If $t \neq 1$, we just take the final population at the previous time step as the initial population at the current time step (see Lines 4 of Algorithm 3). For the other case (i.e., $t = 1$), we randomly produce the initial population by using Algorithm 4, which is explained as follows. Lines 1–3 find the set $arg(U_j)$ that collects the indices of the available server clusters for each client j. Next, the set $arg(U_j)$ is used to construct a population of η chromosomes in Line 5, and then we repair each chromosome to be feasible in Lines 6–21. The idea of the repairing operation is that if Constraint (5) is violated, then we find another available server cluster to serve the violated load; otherwise, we drop the load.

Algorithm 4 INITIALIZE

1: **for** $j = 1$ to n **do**
2: $arg(U_j) \leftarrow$ set of integers which store the indices of the available cluster servers U_j that are linked with client j in the reduced bipartite graph G_t (where the links violating Constraints (6) and (7) have been removed in Algorithm 1)
3: **end for**
4: **for** $p = 1$ to η **do**
5: construct the p-th chromosome $c_p = \langle \sigma_1, \cdots, \sigma_j, \cdots, \sigma_n \rangle$, in which σ_j is a number chosen arbitrarily from $arg(U_j)$ for each $j \in \{1, \cdots, n\}$
6: let o be the set of the positions of 0's in c_p that do not violate our problem constraints
7: **for** $i = 1$ to m **do**
8: scan chromosome c_p to compute the $\sum_{j \in U} x_{ij}^t l_{ij}^t$ value for server cluster i
9: $g_i \leftarrow \sum_{j \in U} x_{ij}^t l_{ij}^t - \sum_{k \in K_i} C_{ik}$
10: let o_i be the set of o associated with cluster server i
11: **while** $g_i > 0$ **do**
12: find the index x such that $\sigma_x = i$ in c_p, $l_{ix} > g_i$, and the gap between l_{ix} and g_i is the smallest
13: **if** there is a position o_y in o that can accommodate l_{ix} and satisfy all the problem constraints **then**
14: swap the values of o_y and σ_x
15: remove o_y in o
16: $g_i \leftarrow g_i - l_{ix}$
17: **else**
18: $\sigma_x \leftarrow 0$
19: **end if**
20: **end while**
21: **end for**
22: **end for**

2) Selection, Crossover: As for the selection operation, it is common that the GA uses the roulette wheel selection. Our GA continues using the roulette wheel selection. Two chromosomes chosen randomly from the selected parental pool are crossovered for generating two new child chromosomes that have to keep some characteristics of its parent chromosomes. The information of chromosomes depends on the fitness function structure. Here, we continue using the conventional one-point crossover operation. Interested readers can be referred to [15] for the details on roulette wheel selection and one-point crossover.

3) Mutation: To prevent our GA from falling into a local optimal solution, we need to change some genes on some chromosomes. We assume that each chromosome has a given probability p_m to mutate. In the mutated chromosome, a nonzero gene σ_x is chosen randomly, and then we randomly find a zero gene σ_y that can accommodate σ_x and do not violate any problem constraint. Then, we swap the values of σ_x and σ_y. These modifications may not necessarily improve the fitness values. However, the worse chromosomes will be eliminated on the selection step. As long as mutation could improve chromosomes, it is still a meaningful step.

4) Repair: After the above evolving operations, the modified chromosomes may become infeasible, and hence, we need to repair them to be feasible. Since the truth of Constraint (4) can always be guaranteed in our design of chromosomes, and the infeasibility of Constraint (6) and Constraint (7) has been filtered in the preprocessing stage of the GA, thus we only need to repair those chromosomes that violate Constraint (5). The repairing operation is referred to Lines 6–21 in Algorithm 4, which have been used in initializing chromosomes.

5) Termination: If the difference of average fitness values between successive generations in the latest ten generations is no grater than 1% of the average of the average fitness values of these ten generations, or the maximum generations are achieved, then our GA stops. After termination, the best chromosome from the latest population is chosen, and its corresponding load assignment is outputted as the final solution.

IV. CONVERGENCE ANALYSIS OF THE GENETIC ALGORITHM WITH IMMIGRANT SCHEME

Although the GA with immigrant scheme was proposed in [21] for coping with dynamic problems, the convergence of the algorithm was never analyzed. Hence, this section applies the concept of Markov chains to the convergence analysis, by the analogy from [29] for analyzing convergence of the GA with local search.

First of all, some notations and definitions for Markov chains are given as follows. A stochastic process $\{\hat{X}_n, n = 0, 1, 2, ...\}$ is called a *Markov chain* if $P\{\hat{X}_{n+1} = j | \hat{X}_n = i\} = P_{ij}$ for any states i, j and $n > 0$, in which $\hat{X}_t = k$ means the process to be in state k at time step t; P_{ij} is a probability depending on only states i and j, not on time step n. Markov chain $\{\hat{X}_n\}$ is called *homogeneous* if $P\{\hat{X}_{m+n} = j | \hat{X}_m = i\} = P_{ij}^{(n)}$, which is the n-step transition probability from state i to state j in $\{\hat{X}_n\}$, i.e., it is not dependent of the former m time steps. If $P_{ij}^{(n)} > 0$ and $P_{ji}^{(n)} > 0$, the two states i and j are said to be accessible to each other, and are in the same *class*. If there is only one class, the Markov chain is said to be *irreducible*. Let p_i denote the probability that the process starts in state i and will ever re-enter state i. If $p_i = 1$, state i is said to be *recurrent*. If state i is recurrent and the expected time starting state i until

the process returns to state i is finite, then state i is said to be *positive recurrent*. If $P_{ii}^{(n)} = 0$ whenever n is not divisible by d (in which d is the largest integer with this property), state i is said to have *period* d. A state with period 1 is said to be *aperiodic*. A state is called *ergodic* if it is positive recurrent and aperiodic. There are two lemmas in [30] as follows:

Lemma 1. *All recurrent states in a finite-state Markov chain are positive recurrent.*

Lemma 2. *For an irreducible ergodic Markov chain,* $\lim_{n\to\infty} P_{ij}^{(n)}$ *exists and is independent of* i*. Furthermore, if let* $\pi_j = \lim_{n\to\infty} P_{ij}^{(n)}, j \geq 0$*, then* π_j *is the unique nonnegative solution of* $\pi_j = \sum_{i=0}^{\infty} \pi_i P_{ij}, j \geq 0$*, subject to* $\sum_{j=0}^{\infty} \pi_j = 1$.

In order to analyze our GA, we continue using the following hypotheses given in [29]: 1) the number of local optimal solutions is finite; 2) each solution can be mapped to a local optimal point by local search. Let H_L be the domain space of our concerned problem, which is expressed as follows: $H_L = \mathcal{G}(x_1^{\#}) \cup \mathcal{G}(x_2^{\#}) \cup \cdots \cup \mathcal{G}(x_L^{\#})$ where all the local optimal points in H_L are denoted by $x_1^{\#}, x_2^{\#}, \cdots, x_L^{\#}$, $L \geq 3$ (without loss of generality), and one of them is the global optimal point; $\mathcal{G}(x_i^{\#}) \subset H_L$ denotes the set that for each $x_i \in \mathcal{G}(x_i^{\#})$ we can use local search to find the $x_i^{\#}$ that started from x_i. Although our GA does not apply local search, the above hypotheses are still suitable for proving correctness of our theorem.

In what follows, we start to discuss the convergence of our GA. Remind that η denotes the number of chromosomes. Let $\overrightarrow{X}(t) = \{X_1(t), X_2(t), \cdots, X_\eta(t)\}$ be the population of the t-th generation of our algorithm. Hence, the state space of the population sequence $\{X(t)\}$ produced by our algorithm is denoted by $H_L^\eta = \mathcal{G}(x_1^{\#}) \cup \mathcal{G}(x_2^{\#}) \cup \cdots \cup \mathcal{G}(x_\eta^{\#})$ where $\eta \leq L$ (with loss of generality). Let B^* be the set of the optimal solutions and f^* be the optimal fitness value. We say that $\{X(t)\}$ *almost surely converges* to B^* as follows:

Definition 1. *The population sequence* $\{X(t)\}$ *is said to* almost surely converge *to* B^* *if the probability* $P\{\overrightarrow{X}(t) \subseteq B^*\}$ *satisfies* $P\{\lim_{t\to\infty}[\overrightarrow{X}(t) \subseteq B^*]\} = 1$.

Our GA can be regarded as a composite mapping from population $X(t)$ to $X(t+1)$: $T = I_r \circ I_e \circ R \circ S$ in which $S : H_L^\eta \to H_L^\eta \times H_L^\eta$ is the mapping for selecting two parents; $R : H_L^\eta \times H_L^\eta \to H_L$ is the mapping for performing crossover and mutation operators; $I_e : H_L \to H_L$ is the mapping for replacing worst chromosomes by elite immigrants, while $I_r : H_L \to H_L$ is the mapping for replacing worst chromosomes by random immigrants.

Our GA can be analyzed by Markov chains as follows:

1) Initialize $\overrightarrow{X}(0)$ randomly.
2) Using roulette wheel selection to select η pairs of parents from $\overrightarrow{X}(t)$ at the t-th time step with the following probability distribution:

$$P\{S(\overrightarrow{X}(t)) = (Y_1^{(k)}, Y_2^{(k)})\} = \begin{cases} \frac{f(Y_1^{(k)\#}) f(Y_2^{(k)\#})}{(\sum_{i=1}^{\eta} f(X_i^{\#}(t)))^2}, & \text{if } Y_1^{(k)}, Y_2^{(k)} \in \overrightarrow{X}(t); \\ 0, & \text{otherwise;} \end{cases}$$

for $k = 1, 2, ..., \eta$. Note that $(Y_1^{(k)}, Y_2^{(k)})$ are the k-th pair of parents selected by roulette wheel, so the probability is composed of two nonzero ratios of $Y_1^{(k)}$ and $Y_2^{(k)}$ over $\overrightarrow{X}(t)$. Hence, the probability is positive.

3) After crossover and mutation, the population reconcile the following probability distribution:

$$P\{R(Y_1^{(k)}, Y_2^{(k)}) = Z^{(k)}\} = \begin{cases} \alpha, & \text{if } Z^{(k)} \in \mathcal{G}(Y_1^{(k)\#}); \\ \beta, & \text{if } Z^{(k)} \in \mathcal{G}(Y_2^{(k)\#}); \\ 1-\alpha-\beta, & \text{if } Z^{(k)} \\ & \quad \in H_L \setminus \mathcal{G}(Y_1^{(k)\#}) \cup \mathcal{G}(Y_2^{(k)\#}); \\ 0, & \text{otherwise;} \end{cases}$$

for $k = 1, 2, ..., \eta$, in which $\alpha, \beta, 1-\alpha-\beta > 0$ by the Lebesque measure theory and $L \geq 3$ [29].

4) The sequence $\{Z^{(k)}\}$ are reordered such that $f(Z^{(1)}) \leq f(Z^{(2)}) \leq ... \leq f(Z^{(\eta)})$. Remind that smaller fitness f is better in our setting. After adding elite immigrants, the population reconcile the following distribution:

$$P\{I_e(Z) = W^{(k)}\} = \begin{cases} \gamma_1, & \text{if } W^{(k)} \in \{Z^{(1)}, ..., Z^{(r_e\eta)}\}; \\ \gamma_2, & \text{if } W^{(k)} \in \{Z^{(r_e\eta+1)}, ..., Z^{(\eta-r_e\eta-1)}\}; \\ 0, & \text{otherwise;} \end{cases}$$

for $k = 1, 2, ..., \eta$.

5) The sequence $\{W^{(k)}\}$ are reordered such that $f(W^{(1)}) \leq f(W^{(2)}) \leq ... \leq f(W^{(\eta)})$. After adding random immigrants, the population reconcile the following probability distribution:

$$P\{I_r(W^{(k)}) = X_k(t+1)\} = \begin{cases} \delta, & \text{if } X_k(t+1) \in \mathcal{G}(W^{(k)\#}); \\ 1-\delta, & \text{if } X_k(t+1) \in H_L \setminus \mathcal{G}(W^{(k)\#}); \\ 0, & \text{otherwise;} \end{cases}$$

for $k = 1, 2, ..., \eta$.

It is worthy mentioning the difference between our analysis and that in [29] as follows. On the Markov chain, in addition to mappings S and R proposed originally in [29], we additionally consider mappings I_e and I_r, which are designed for immigrant scheme. The correctness of the following lemma can be shown by a similar proof with in [29]:

Lemma 3. *The population sequence* $\{\overrightarrow{X}(t)\}$ *generated by our GA is a finite, homogeneous, irreducible, and aperiodic Markov chain if all the nonzero probabilities in 1) – 5) are positive and all independent of* t.

As a result, we can show the following theorem:

Theorem 1. *Let* $f(x)$ *be a real continuous function and* H_L *be an* l*-dimensional box constraint. By using the roulette wheel selection, crossover, mutation, elite immigrants, and random immigrants in our GA, then the* $\{\overrightarrow{X}^*(t)\}$ *with an elitist strategy almost surely converges to the set of the optimal solution set* B^*.

Proof. (Sketch) By Lemmas 1 – 3 and the analogy with [29], it suffices to show that the nonzero probabilities 4) and 5) are positive and all independent of t.

For similarity, we assume that $f(x) \geq 0$ for all H_L (otherwise, a sufficiently large constant can be added to $f(x)$) and $S = [0,1]^l$. Hence, it is obvious that the Lebesque measure $m(H_L) = 1$ and $\sum_{i=1}^{L} m(\mathcal{G}(X_i^{\#})) = 1$. For the process of 4), the variable γ_1 and γ_2 are represented as follows:

$$\gamma_1 = 2/\eta,$$
$$\gamma_2 = 1/\eta.$$

The above equations hold because elite (better) chromosomes in the η chromosomes are duplicated after elite immigrants join the population, while the other chromosomes are unchanged. In the process of 5), the probability of $X_k(t+1)$ belongs to $\mathcal{G}(W^{(k)\#})$ or $H_L \setminus \mathcal{G}(W^{(k)\#})$ depends on the Lebesque measure of $m(\mathcal{G}(W^{(k)\#}))$. So we can know:

$$\delta = m(\mathcal{G}(W^{(k)\#}))$$

and $1 - \delta > 0$ That it, the nonzero probabilities in 4) and 5) are positive and all independent of t, as required. □

V. Implementation and Experimental Results

This section first explains how the data used in experiments were generated and the experimental environment, and then gives the experimental results of a variety of cases.

A. Data and Simulation Environment

We consider an instance with 20 server clusters ($m = 20$) and 100 clients ($n = 100$). The weight of each link is bounded in the range $[0, 5]$ in general. That is, the normalizing factor of the first term in Objective (3) is $5 \cdot 100 = 500$, while that of the second term is 100. If the link is infeasible, its weight is set 1000, which is viewed as infinity in our experiments.

In our experiments, unless otherwise described in the rest of this paper, our GA algorithm applies the parameter settings in Table I, in which there are 200 generations at most; there are 50 chromosomes in a generation; the time period between two time steps is the time taken by 20 generations of the main loop of the GA algorithm. That is, clients move at each time step, and their corresponding criteria are measured at every 20 generations. In addition, after a lot of tests, $\lambda = 0.7$ is chosen.

TABLE I
Parameters used in simulation.

Parameter	Value
Number of server clusters (m)	20
Number of clients (n)	100
Maximal weight ($\max_{i \in U}\{w_{ij}^t\}$)	5
Weight of first objective (λ)	0.1
Maximal number of generations (τ)	200
Number of generations between two time steps	20
Number of chromosomes (η)	50
Size of parental pool ($\lvert\mathcal{Q}_i^t\rvert$)	50
Crossover rate (p_c)	0.5
Mutation rate (p_m)	0.02
Rate of elite immigrants (r_e)	0.2
Rate of random immigrants (r_r)	0.2

Our simulation was tested on an Intel Core i7-3770 CPU @ 3.40 GHz with 16 GB memory. The average running time for determining a placement of an instance (i.e., 20 generations) is about 0.0005 seconds. It implies that our GA has the ability to efficiently cope with the CMS-dynMLB problem.

B. Experimental Results

To the best of our understanding, there were no previous works that studied our concerned problem. As a result, we conduct a comprehensive experimental analysis on adjustment of parameters. First, in order to observe the convergence of the best cost values in our GA method, we plot the best cost values versus the number of generations of our GA under a variety of parameters in Figure 2, from which each plot is convergent to a fixed value, which implies that our GA has the ability to make the solutions to be convergent.

Figure 2(a) gives plots for different numbers of server clusters (m), in which it is reasonable that more server clusters (resources) converge faster. In order to make the problem instance not easy to be solved, we choose $m = 20$, by which the solution converges at about 6-th generation. Similarly, from Figures 2(b) and 2(c), the case of $\lambda = 1$ and $\eta = 50$ have nontrivial plots, and hence we apply the settings. As for crossover and mutation rates, we choose $p_c = 0.5$ and $p_m = 0.02$, because those settings have the best performance from Figures 2(d)–2(f). As for immigrant scheme, we choose $r_e = 0.2$ and $r_r = 0.2$, in which $r_e = 0.2$ is due to its better performance from Figure 2(g). Although $r_r = 0.2$ has worse performance in Figures 2(h) and 2(i), its performance in dynamic scenarios is better.

In order to demonstrate the ability of our approach to adapt the time changes (where we suppose that the topology graph changes in each 20 generations), we run 200 generations of our GA on the test instance in a dynamic scenario, and its plots of best cost values versus the iteration number under three different m values are given in Figure 3. The dynamic scenario assumes that all of the clients change their locations in each 20 generations in Figure 3, from which we observe that every time when clients change their location in each 20 generations, the cost value goes to a large value, and later converges by our GA approach. In addition, we also observe that from (a) to (c) more clients provide more resources, so that the plots turn out to be a plat region more quickly.

VI. Conclusion

A genetic algorithm approach for optimizing the dynamic multi-service load balancing in cloud-based multimedia system (CMS-dynMLB) has been proposed and implemented. The main difference of our model from previous models is that we consider a practical multi-service dynamic scenario in which at different time steps, clients can change their locations, and each server cluster only handles a specific type of multimedia tasks, so that two performance objectives are optimized at the same time. The main features of this paper include not only the proposal of a mathematical formulation of the CMS-dynMLB problem but also a theoretical analysis for the algorithm convergence. Detailed simulation has also been conducted to show the performance of our GA approach.

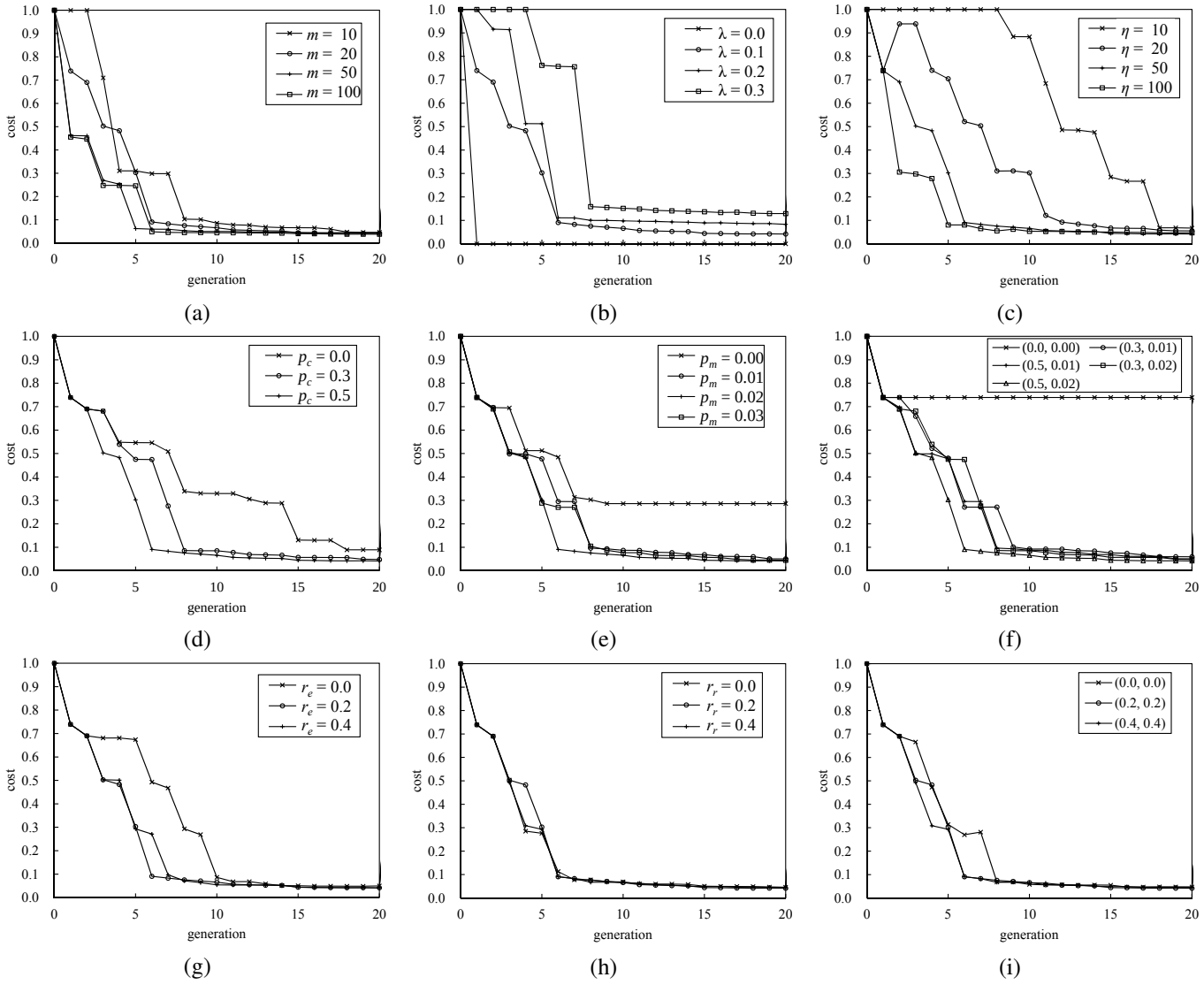

Fig. 2. Static scenarios under different (a) numbers of server clusters, (b) λ values, (c) η values, (d) P_c values, (e) P_m values, (f) combinations of P_c and P_m values, (g) r_e values, (h) r_r values, and (i) combinations of r_e and r_r.

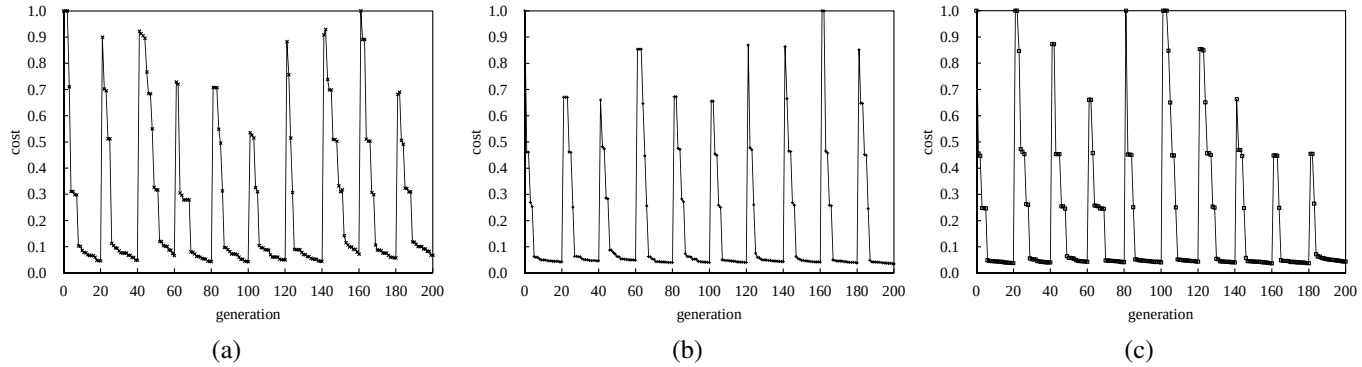

Fig. 3. Dynamic scenarios under different m values: (a) 20, (b) 50, and (c) 100.

ACKNOWLEDGEMENTS

The authors thank the anonymous referees for comments that improved the content as well as the presentation of this paper. This work has been supported in part by NSC 101-2219-E-009-025 and NSC 101-2221-E018-004, Taiwan.

REFERENCES

[1] W. Zhu, C. Luo, J. Wang, and S. Li, "Multimedia cloud computing: An emerging technology for providing multimedia services and appli-

cations," *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 59–69, 2011.

[2] C.-F. Lai, Y.-M. Huang, and H.-C. Chao, "DLNA-based multimedia sharing system over OSGI framework with extension to P2P network," *IEEE Systems Journal*, vol. 4, no. 2, pp. 262–270, 2010.

[3] W. Hui, H. Zhao, C. Lin, and Y. Yang, "Effective load balancing for cloud-based multimedia system," in *Proceedings of 2011 International Conference on Electronic & Mechanical Engineering and Information Technology*. IEEE Press, 2011, pp. 165–168.

[4] C.-Y. Chen, H.-C. Chao, S.-Y. Kuo, and K.-D. Chang, "Rule-based intrusion detection mechanism for IP multimedia subsystem," *Journal of Internet Technology*, vol. 9, no. 5, pp. 329–336, 2008.

[5] L. J. Wu, A. E. AL Sabbagh, K. Sandrasegaran, M. Elkashlan, and C. C. Lin, "Performance evaluation on common radio resource management algorithms," in *Proceedings of 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA 2010)*. IEEE Press, 2010, pp. 491–495.

[6] R. Yavatkar, D. Pendarakis, and R. Guerin, "A framework for policy based admission control," Internet Requests for Comments, RFC Editor, RFC 2753, 2000.

[7] D. Niyato and E. Hossain, "Integration of WiMAX and WiFi: Optimal pricing for bandwidth sharing," *IEEE Communication Magazine*, vol. 45, no. 5, pp. 140–146, 2007.

[8] C.-Y. Chang, T.-Y. Wu, C.-C. Huang, A. J.-W. Whang, and H.-C. Chao, "Robust header compression with load balance and dynamic bandwidth aggregation capabilities in WLAN," *Journal of Internet Technology*, vol. 8, no. 3, pp. 365–372, 2007.

[9] J. Sun, X. Wu, and X. Sha, "Load balancing algorithm with multi-service in heterogeneous wireless networks," in *Proceedings of 6th International ICST Conference on Communications and Networking in China (ChinaCom 2011)*. IEEE Press, 2011, pp. 703–707.

[10] H. Son, S. Lee, S.-C. Kim, and Y.-S. Shin, "Soft load balancing over heterogeneous wireless networks," *IEEE Transactions on Vehichular Technology*, vol. 57, no. 4, pp. 2632–2638, 2008.

[11] L. Zhou, H.-C. Chao, and A. V. Vasilakos, "Joint forensics-scheduling strategy for delay-sensitive multimedia applications over heterogeneous networks," *IEEE Journal on Selected Areas of Communications*, vol. 29, no. 7, pp. 1358–1367, 2011.

[12] X. Nan, Y. He, and L. Guan, "Optimal resource allocation for multimedia cloud based on queuing model," in *Proceedings of 2011 IEEE 13th International Workshop on Multimedia Signal Processing (MMSP 2011)*. IEEE Press, 2011, pp. 1–6.

[13] M. Garey and D. Johnson, *Computers and Intractability - A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[14] S. Kirkpatrik, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[15] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*. IEEE Press, 1995, p. 1942V1948.

[17] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE Press, 1998, pp. 69–73.

[18] X. Zhang, S. Hu, D. Chen, and X. Li, "Fast covariance matching with fuzzy genetic algorithm," *IEEE Transactions on Industrial Engineering*, vol. 8, no. 1, pp. 148–157, 2012.

[19] W. Ip, D. Wang, and V. Cho, "Aircraft ground service scheduling problems and their genetic algorithm with hybrid assignment and sequence encoding scheme," *IEEE Systems Journal*, 2012, to appear.

[20] F. Gonzalez-Longatt, P. Wall, P. Regulski, and V. Terzija, "Optimal electric network design for a large offshore wind farm based on a modified genetic algorithm approach," *IEEE Systems Journal*, vol. 6, no. 1, pp. 164–172, 2012.

[21] H. Cheng and S. Yang, "Genetic algorithms with immigrants schemes for dynamic multicast problems in mobile ad hoc networks," *Engineering Applications of Artificial Intelligence*, vol. 23, no. 5, pp. 806–819, 2010.

[22] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads," in *Proceedings of 2010 IEEE 3rd International Conference on Cloud Computing*. IEEE Press, 2010, pp. 228–235.

[23] K.-P. Chow and Y.-K. Kwok, "On load balancing for distributed multiagent computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 8, pp. 787–801, 2002.

[24] X. Qin, H. Jiang, A. Manzanares, X. Ruan, and S. Yin, "Communication-aware load balancing for parallel applications on clusters," *IEEE Transactions on Computers*, vol. 59, no. 1, pp. 42–52, 2010.

[25] A. Y. Zomaya and Y.-H. Teh, "Observations on using genetic algorithms for dynamic load-balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 9, pp. 899–911, 2001.

[26] Y.-M. Huang, M.-Y. Hsieh, H.-C. Chao, S.-H. Hung, and J. H. Park, "Pervasive, secure access to a hierarchical-based healthcare monitoring architecture in wireless heterogeneous sensor networks," *IEEE Journal on Selected Areas of Communications*, vol. 27, no. 4, pp. 400–411, 2009.

[27] L. Yang and M. Guo, *High-performance Computing: Paradigm and Infrastructure*. John Wiley and Sons, 2006.

[28] T.-Y. Wu, H.-C. Chao, and C.-Y. Huang, "A survey of mobile IP in cellular and mobile ad-hoc network environments," *Ad Hoc Networks Journal*, vol. 3, no. 3, pp. 351–370, 2005.

[29] Q. Yuan, F. Qian, and W. Du, "A hybrid genetic algorithm with the Baldwin effect," *Information Sciences*, vol. 180, no. 5, pp. 640–652, 2010.

[30] S. Ross, *Introduction to Probability Models*, 10th ed. Academic Press, 2009.