Genetic Algorithm with Range Selection Mechanism for Dynamic Multiservice Load Balancing in Cloud-Based Multimedia System

Michael Sadgun Rao Kona, K.Purushottama Rao
Dept. of Information Technology, LBRCE, JNTUK University, Andhra Pradesh, India

Abstract
Consider a centralized hierarchical cloud-based multimedia system (CMS) which has a resource manager, cluster heads, and server clusters. Resource manager accepts clients’ requests for multimedia service tasks and allocates server clusters based on the task characteristics. Cluster head distributes the task to the servers within its server cluster. An effective genetic algorithm with an immigrant scheme is proposed for load balancing that spreads the multimedia service task load on servers with the minimal cost for transmitting multimedia data between server clusters and clients, by not violating the maximal load limit of each server cluster. Dynamic multiservice scenario is considered in which each server cluster only handles a specific type of multimedia operation, and every client requests a different type of multimedia service at a different time.

Keywords
Cloud computing, load balancing, genetic algorithm and multimedia system.

I. Introduction
Cloud-based multimedia system (CMS) [3] gained momentum because there is huge number of users’ demands for various multimedia computing and storage services through the Internet at the same time [4], [5]. It provides infrastructure, platforms, and software as service to large number of clients at the same time in order to store and process their multimedia application data in a distributed manner by meeting different multimedia QoS requirements through the Internet.
Cloud computing is strongly required in most multimedia applications with huge computation and for mobile devices which are power constrained.
In general, a cloud service provider offers required facilities to clients which take less cost to request, process and to compute multimedia services. So, multimedia applications are processed on powerful cloud servers with less cost because client should only pay for the utilized resources by the time.
We are considering a centralized hierarchical CMS from [3] 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.

Operation of CMS
Resource manager assigns client requested multimedia service task to different server clusters according to the characteristics of the requested tasks. Cluster head will distribute the assigned task to some server within the server cluster.
The load of each server cluster affects the performance of the whole CMS. Resource manager of the CMS is required to distribute the task load across server clusters, and hence, it is important to be able to cope with load balancing in the CMS.
Services offered by CMS are generating, editing, processing, and searching a variety of multimedia data, e.g., hypertext, images, video, audio, graphics, and so on with various requirements for the functions provided by the CMS (storage, central processing unit, and graphics processing unit clusters), e.g., the requirement for QoS of html webpage services is looser than that of video streaming services.

II. Problem Description
Actually this load balancing problem for the CMS is from [1], and we are extending their model with different selection method in GA.

A. System Overview
CMSs can be classified into two categories: centralized and decentralized. We are considering a centralized CMS as illustrated in Fig. 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, whenever it receives requests from client 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 terms 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.

B. Problem Formulation

We are using the same CMS problem formulation as in [1], in which time is divided into different time steps. At the tth time step, the CMS is modeled as a complete weighted bipartite graph

\[ G_t = (U, V, E, \varphi, \psi, q, r', w') \]

in which U is the set of vertices for server clusters, V is the set of vertices for 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 \( j \in V \). \( \varphi \) is a function used to represent that server cluster \( i \) can only cope with multimedia tasks of type \( \varphi_i \), \( \psi \) is a function used to represent that client \( j \) requests the multimedia service of type \( \psi_j \) at the tth time step. \( q \) is a function used to represent that server cluster \( i \) can provide the multimedia service of QoS \( q_i \), \( r \) is a function used to represent that client \( j \) requests the multimedia service of QoS requirement \( r_j \) at the tth time step and finally \( w \) is the weight function associated with edges, in which \( w_{ij} \) denotes the wt value that represents the cost for transmitting multimedia data between server cluster i and client j at the tth time step, which is defined as follows:

\[ w_{ij} = \begin{cases} \infty, & \text{if } d_{ij} = \infty \text{ or } \varphi_i \neq \psi_j \\ w_{ij}, & \text{otherwise} \end{cases} \]  

(1)

where

\( d_{ij} \) is the network proximity between server cluster i and client j;

\( l_{ij} \) is the traffic load of the link between server cluster i and client j that is defined as follows:

\[ l_{ij} = \sum_{k \in K_i} u_{kij} C_k \]  

(2)

where \( K_i \) is the servers’ set in server cluster i; \( u_{kij} \) is the server utilization ratio of server k in server cluster i due to client j, and \( C_k \) is its capacity. Note that the proximity \( d_{ij} \) between server cluster i and client j in (1) is required to be measured at every time step due to dynamic change of network topology. Proximity between nodes is calculated with landmark order and Binning scheme.

This paper uses the same linear programming formulation used in [1] for every tth time step with constraints. Each client only allows at most one link to be assigned (Constraint (4)), the utilized capacity of each server cluster cannot exceed its capacity at the tth time step (Constraint (5)), multimedia service type requested by each client j is consistent with that provided by server cluster i (Constraint (6)) and each client j requests the multimedia server of the QoS no more than that offered by server cluster i (Constraint (7)).

Dynamic Multiservice Load Balancing In Cms (Cms-Dynmlb): \( G_t=(U, V, E, \varphi, \psi', q, r', w') \) is the bipartite graph for CMS with m servers and n clients for each time step \( t=1,2,\ldots \). This paper uses Range selection method for genetic algorithm (GA) with immigrant scheme [1] for better results in solving the problem.
if \( r > 0 \) then

- generate and evaluate \( r \cdot \eta \) random immigrants

end if

- replace the worst chromosomes in \( P_t \) with the above

elite and random immigrants

\[ P_{t+1} \leftarrow P_t \]

\[ t \leftarrow t+1 \]

end while

output the best found chromosome as the solution at the

\( t^{th} \) time step

Algorithm 4 Initialize

for \( j = 1 \) to \( n \) do

\[ \arg(U_j) \leftarrow \text{set of integers which store the indices of the available} \]

cluster servers \( U_j \) that are linked with client \( j \) in the reduced

bipartite graph \( G \), (where the links violating Constraints (6) and

(7) have been removed in Algorithm 1)

end for

for \( p = 1 \) to \( \eta \) do

- construct the \( p^{th} \) chromosome \( c_p = \sigma_1, \ldots, \sigma_j, \ldots, \sigma_n \), in

which \( \sigma_j \) is a number chosen arbitrarily from \( \arg(U_j) \) for each \( j \in \{1, \ldots, n\} \)

let \( o \) be the set of the positions of 0’s in \( c_p \) that do not violate our

problem constraints

for \( i = 1 \) to \( m \) do

- scan chromosome \( c_p \) to compute the \( j \in U \) \( x_{ij}^p \) value

for server cluster \( i \)

\[ g_i \leftarrow \sum_{j \in U} x_{ij}^p \times g_j \]

let \( o_i \) be the set of \( o \) associated with cluster server \( i \)

while \( g_i > 0 \) do

- find the index \( x \) such that \( \sigma_j = i \) in \( c_p \), \( l_{ix} > g_i \), and the gap

between \( l_x \) and \( g_i \) is the smallest if there is a position \( o_j \) in \( o \)

that can accommodate \( l_x \) and satisfy all the problem constraints

- swap the values of \( o_y \) and \( \sigma_x \)

- remove \( o_x \) in \( o \)

\[ g_i \leftarrow g_i - l_x \]

else

\[ \sigma_x \leftarrow 0 \]

end if

end while

end for

\textbf{end for}

\section{III. Our Genetic Algorithm}

\textbf{A. Our Algorithm}

GA is based on imitating the evolutional behavior of a population

of chromosomes (each 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 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 crossed over 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.

At each time step \( t \), Algorithm 1 iterates on \( t \) to reallocate

the network load assignments so as to adapt to the time change. First,

Line 2 constructs a complete weighted bipartite graph \( G \). Line

3 removes the links in \( G \), violating (6) and (7). 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 Algorithm 3

to compute our final load assignment solution \( \{x_{ij}^t\} \).

In algorithm 2, Line 1 considers each client \( j \in V \) to compute its

weight \( w_{ij}^t \) with each server cluster \( i \). We are using distributed

binning scheme to calculating the proximity, so, 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 includes the server

clusters connected to client \( j \), Lines 6 and 7 calculate the landmark

distance \( l_i \) of server cluster \( i \). In Lines 8–14, if \( l_j = l_i \) (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 \( \infty \).

With the values of \( \{l_{ij}^t\} \) and \( \{w_{ij}^t\} \), we are ready to apply the

GA to computing the optimal load assignment in Algorithm 3. The GA

runs differently based on two parameters: the input bipartite

graph \( G \) and the input time step \( t \). Recall that \( G \) may not be a

complete graph any longer, because the links that violate some

problem constraints have been removed in Line 3 of Algorithm 1.

Let \( \eta \) and \( \tau \) denote the size of a population and the number of

generations, respectively. In Lines 1–5, the initial population of

\( \eta \) chromosomes is 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 that satisfies the remaining two constraints that we did

not consider yet (i.e., (4) and (5)). Otherwise (i.e., \( t \neq 1 \), which

implies that there existed the final population \( P_{t-1} \) of the \( (t-1) \)

a time step), Line 4 uses \( P_{t-1} \) as the initial population at the \( t^t \)

time step Subsequently, the while loop in Lines 7–23 repeats at most \( \tau \) iterations, each of which produces a population \( P_t \), of

chromosomes, i.e., the next population at the \( t^t \) time step. Line 8

selects a number of chromosomes from the population \( P_t \) as the

parental pool \( Q_t \), Lines 9 and 10 perform crossover and mutation

operators to the population \( P_t \) with probabilities \( p_c \) and \( p_m \),

respectively. 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

[6]. 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 \( P_t \) 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.

\textbf{B. Basic Elements of Our GA}

To use GA in order 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: A population consists of a number of chromosomes,

and the number of chromosomes depends on the given initial

population size.
2) Chromosome: A solution for the CMS-dynMLB problem consists of all the indicator variables \( \{ x_{ij}^t \mid \forall i \in U, j \in V \} \). The solution for indicator variables \( \{ x_{ij}^t \} \) is encoded as a sequence of fractional numbers of length \( n \{ \sigma_1, \sigma_2, \cdots, \sigma_n \} \), where \( \sigma_i \in \{0, 1, 2, \cdots, m\} \) represents the link assignment of client \( j \) with server cluster \( k \). If \( \sigma_i = 0 \), then each \( x_{ij}^t \) is 0 for any \( i \in U \), i.e., client \( j \) is not linked at the \( t \)th time step. Otherwise, \( \sigma_i = k \neq 0 \), meaning that \( x_{ij}^t = 1 \) and \( x_{ij}^t = 0 \) for any \( i \neq k \), i.e., client \( j \) is linked to server cluster \( k \) uniquely.

3) Fitness Function: Fitness function is the measure for determining which chromosomes are better or worse. We let the objective (3) of our concerned problem as our fitness function as follows: 

\[
f(X(t)) = \frac{\lambda \cdot \sum_{i \in V} \sum_{j \in U} x_{ij} w_i}{W^{max}} + (1 - \lambda) \cdot (1 - \sum_{j \in V} \sum_{i \in U} x_{ij} / |V|)
\]

where \( X(t) \) is the profile of \( \{ x_{ij}^t \} \).

C. Main Components of Our GA

1) Initialization: In Lines 1–5 of Algorithm 3, we use two cases of our concerned problem as our fitness function as follows: 

- \( f \) values \( f \) are calculated in Lines 1–3, which find the set \( \arg(U_j) \) which 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 \( \eta \) 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 (5) is violated, then we find another available server cluster to serve the violated load; otherwise, we drop the load.

2) Selection, Crossover: The range selection procedure is used. Each individual in the population is assigned a numerical rank based on their fitness, and selection is based on these ranking rather than absolute differences in fitness. The advantage of this method is that it can prevent very fit individuals to gain dominance at first at the expense of the less fit, which would reduce the genetic diversity of the population and could hamper the search for an acceptable solution [7]. One-point crossover operation is used in which two selected chromosomes are crossovered for generating two new child chromosomes in order to keep some characteristics of its parent chromosomes.

3) Mutation: Changing some genes on some chromosomes. Let \( pm \) be given probability to mutate. In the mutated chromosome, a nonzero gene \( \sigma_i \) is chosen randomly, and then we randomly find a zero gene \( \sigma_i \) that can accommodate \( \sigma_i \) and do not violate any problem constraint. Then, we swap the values of \( \sigma_i \) and \( \sigma_j \).

4) Repair: The modified chromosomes may become infeasible, and hence repair is required. 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 not greater than 1% 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. Implementation and Experimental Results

A. Implementation

CloudSim 3.0.3[2] is used for simulation with java as programming language. CloudSim has support for modeling and simulation of large scale Cloud computing environments, including data centers, on a single physical computing node. It has necessary classes in java for simulating every entity that involved in our project like Datacenter, Host, VM, Cloudlet, DatacenterBroker etc. The CloudSim toolkit supports both system and behaviour modelling of Cloud system components. It has availability of a virtualization engine that aids in creation and management of multiple, independent, and co-hosted virtualized services on a data center node.

Our simulation was tested on an Intel Core i3-2328M CPU at 2.20 GHz with 4-GB memory.

B. Results

With Range selection mechanism implementation in existing genetic algorithm for balancing dynamic multimedia load in cloud based CMS there is a little betterment.

V. Conclusion

A genetic algorithm approach for optimizing the CMSdynMLB was proposed and implemented. The main difference in our model from previous models is that we considered a Range selection mechanism in genetic algorithm along with practical multiservice dynamic scenario in which at different time steps, clients can change their locations.

References


Author’s Profile


K.Purushottam Rao is presently working as Assistant Professor in Dept. of Information Technology, Lakireddy Balireddy College of Engineering, Mylavaram.