© 2015 IJSRSET | Volume 1 | Issue 2 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Overload Avoidance for Dynamic Virtual Machine Resource Allocation Environment

Pillapakam Sridharan Srivatsan, M Manimaran, V Manikandan, M. Murugesan
Dhanalakshmi College of Engineering, Chennai, Tamilnadu, India

ABSTRACT

Cloud computing allows business customers to scale up and down their resource usage based on needs. Many of the touted gains in the cloud model come from resource multiplexing through virtualization technology. In this paper, we present a system that uses virtualization technology to allocate data center resources dynamically based on application demands and support green computing by optimizing the number of servers in use. We introduce the concept of "skewness" to measure the unevenness in the multidimensional resource utilization of a server. By minimizing skewness, we can combine different types of workloads nicely and improve the overall utilization of server resources. We develop a set of heuristics that prevent overload in the system effectively while saving energy used. Trace driven simulation and experiment results demonstrate that our algorithm achieves good performance.
Keywords: multi cloud storage, cloud user, skewness, disaster recovery, reencryption, Green Computing, CMS QoS, TTP, CPDP

I. INTRODUCTION

With the advancement of cloud technology, the usage of multi cloud server has been constantly increasing for easy way of computation and resource allocation. Even though there are many advantages of using the multi cloud server, there are also some disadvantages in the resource allocation and sharing. To overcome this disadvantage we implement a new algorithm "Skewness Algorithm" involving the concepts of Green Computing.
The multi cloud server 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., audio/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 computational results.

This paper considers a centralized hierarchical CMS 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.

Objective

We present a system that uses virtualization technology to allocate data center resources dynamically based on application demands and support green computing by optimizing the number of servers in use.

We introduce the concept of "skewness" to measure the unevenness in the multidimensional resource utilization of a server. By minimizing skewness, we can combine different types of workloads nicely and improve the overall utilization of server resources.

II. METHODS AND MATERIAL

The following methodologies have been followed. The brief explanation is as follows.

A. Multi Cloud Storage

Distributed computing is used to refer to any large collaboration in which many individual personal computer owners allow some of their computer's processing time to be put at the service of a large problem. In our system the each cloud admin consist of data blocks. The cloud user uploads the data into multi-cloud. Cloud computing environment is constructed based on open architectures and interface; it has the capability to incorporate multiple internal and/or external cloud services together to provide high interoperability. We call such a distributed cloud environment as a multi- cloud. A multi-cloud allows clients to easily access his/her resources remotely through interfaces.

B. Data Integrity

Data Integrity is very important in database operations in particular and Data warehousing and Business intelligence in general. Because Data Integrity ensured that data is of high quality, correct, consistent and accessible.

C. Cooperative PDP

Cooperative PDP (CPDP) schemes adopting zero-knowledge property and three-layered index hierarchy, respectively. In particular efficient method for selecting the optimal number of sectors in each block to minimize the computation costs of clients and storage service providers. Cooperative PDP (CPDP) scheme without compromising data privacy based on modern cryptographic techniques.

D. Third Party Auditor

Trusted Third Party (TTP) who is trusted to store verification parameters and offer public query services for these parameters. In our system the Trusted Third Party, view the user data blocks and uploaded to the distributed cloud. In distributed cloud environment each cloud has user data blocks. If any modification tried by cloud owner an alert is send to the Trusted Third Party.

E. Cloud User

The Cloud User who has a large amount of data to be stored in multiple clouds and have the permissions to access and manipulate stored data. The User's Data is converted into data blocks. The data block is uploaded to the cloud. The TPA views the data blocks and Uploaded in multi cloud. The user can update the uploaded data. If the user wants to download their files, the data's in multi-cloud is integrated and downloaded.

F. Disaster Recovery

Back up a file system to cloud storage, using a least-common-denominator cloud interface, thus support many kinds of cloud services. It uses only one cloud to maintain one backup, and focuses on the mechanism in local file system, not the cloud platform. Wood proposed a new cloud service model, i.e., disaster recovery as a cloud service, which leverages the virtual platforms in cloud computing to provide data disaster recovery service. They created a disaster recovery cloud model for web site applications which illustrated that data backup built on top of cloud resources can greatly reduce the cost of data disaster recovery for corporations. However, they didn't study on how to further improve the service quality using multiple clouds.

G. Reencryption

In this paper, we solve this problem by proposing a time-based re-encryption scheme, which enables the cloud servers to automatically re-encrypt data based on their internal clocks. Our solution is built on top of a new encryption scheme, attribute-based encryption, to allow fine-grain access control, and does not require perfect clock synchronization for correctness.

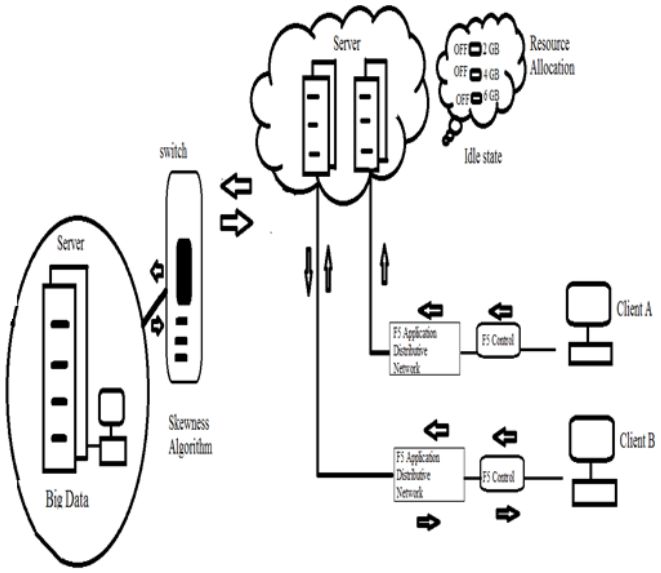

Figure 1: System Architecture

H. Algorithm

Genetic algorithm

In the field of artificial intelligence, a **genetic algorithm (GA)** is a search heuristic that mimics the process of natural selection. This heuristic (also sometimes called a meta heuristic) is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

Genetic algorithms find application in bioinformatics, phylogenetic, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics, pharmacometrics and other fields.

Methodology

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration is called a *generation*. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A typical genetic algorithm requires:

1. A genetic representation of the solution domain,
2. A fitness function to evaluate the solution domain.

A standard representation of each candidate solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

Initialization

The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the *search space*). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

Selection

During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a *fitness-based* process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the former process may be very time-consuming.

The fitness function is defined over the genetic representation and measures the *quality* of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The *fitness* of the solution is the sum of values of all objects in the knapsack if the representation is valid or 0 otherwise.

In some problems, it is hard or even impossible to define the fitness expression; in these cases, a simulation may be used to determine the fitness function value of a phenotype (e.g. computational fluid dynamics is used to determine the air resistance of a vehicle whose shape is encoded as the phenotype), or even interactive genetic algorithms are used.

Genetic operators

The next step is to generate a second generation population of solutions from those selected through a combination of genetic operators: crossover (also called recombination), and mutation.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more "biology inspired", some research suggests that more than two "parents" generate higher quality chromosomes.

These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions. These less fit solutions ensure genetic diversity within the genetic pool of the parents and therefore ensure the genetic diversity of the subsequent generation of children.

Opinion is divided over the importance of crossover versus mutation. There are many references in Fogel (2006) that support the importance of mutation-based search.

Although crossover and mutation are known as the main genetic operators, it is possible to use other operators such as regrouping, colonization-extinction, or migration in genetic algorithms.

It is worth tuning parameters such as the mutation probability, crossover probability and population size to find reasonable settings for the problem class being worked on. A very small mutation rate may lead to genetic drift (which is non-ergodic in nature). A recombination rate that is too high may lead to premature convergence of the genetic algorithm. A mutation rate that is too high may lead to loss of good solutions, unless elitist selection is employed.

Termination

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria.
- Fixed number of generations reached.
- Allocated budget (computation time/money) reached.
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations

no longer produce better results
- Manual inspection
- Combinations of the above

The building block hypothesis

Genetic algorithms are simple to implement, but their behavior is difficult to understand. In particular it is difficult to understand why these algorithms frequently succeed at generating solutions of high fitness when applied to practical problems. The building block hypothesis (BBH) consists of:

- A description of a heuristic that performs adaptation by identifying and recombining "building blocks", i.e. low order, low defining-length schemata with above average fitness.
- A hypothesis that a genetic algorithm performs adaptation by implicitly and efficiently implementing this heuristic.

Goldberg describes the heuristic as follows:

"Short, low order, and highly fit schemata are sampled, recombined [crossed over], and resampled to form strings of potentially higher fitness. In a way, by working with these particular schemata [the building blocks], we have reduced the complexity of our problem; instead of building high-performance strings by trying every conceivable combination, we construct better and better strings from the best partial solutions of past samplings.

"Because highly fit schemata of low defining length and low order play such an important role in the action of genetic algorithms, we have already given them a special name: building blocks. Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks."

Limitations

There are limitations of the use of a genetic algorithm compared to alternative optimization algorithms:

Repeated fitness function evaluation for complex problems is often the most prohibitive and limiting segment of artificial evolutionary algorithms. Finding the optimal solution to complex high-dimensional, multimodal problems often requires very expensive fitness function evaluations. In real world problems such as structural optimization problems, a single function evaluation may require several hours to several days of complete simulation. Typical optimization methods can not deal with such types of problem. In this case, it may be necessary to forgo an exact evaluation and use an approximated fitness that is computationally efficient. It is apparent that amalgamation of approximate models may be one of the most promising approaches to convincingly use GA to solve complex real life problems.

Genetic algorithms do not scale well with complexity. That is, where the number of elements which are exposed to mutation is large there is often an exponential increase in search space size. This makes it extremely difficult to use the technique on problems such as designing an engine, a house or plane. In order to make such problems tractable to evolutionary search, they must be broken down into the simplest representation possible. Hence we typically see evolutionary algorithms encoding designs for fan blades instead of engines, building shapes instead of detailed construction plans, airfoils instead of whole aircraft designs. The second problem of complexity is the issue of how to protect parts that have evolved to represent good solutions from further destructive mutation, particularly when their fitness assessment requires them to combine well with other parts.

The "better" solution is only in comparison to other solutions. As a result, the stop criterion is not clear in every problem.

In many problems, GAs may have a tendency to converge towards local optima or even arbitrary points rather than the global optimum of the problem. This means that it does not "know how" to sacrifice short-term fitness to gain longer-term fitness. The likelihood of this occurring depends on the shape of the fitness landscape: certain problems may provide an easy ascent towards a global optimum, others may make it easier for the function to find the local optima. This problem may be alleviated by using a different fitness function,

increasing the rate of mutation, or by using selection techniques that maintain a diverse population of solutions, although the No Free Lunch theorem proves that there is no general solution to this problem. A common technique to maintain diversity is to impose a "niche penalty", wherein, any group of individuals of sufficient similarity (niche radius) have a penalty added, which will reduce the representation of that group in subsequent generations, permitting other (less similar) individuals to be maintained in the population. This trick, however, may not be effective, depending on the landscape of the problem. Another possible technique would be to simply replace part of the population with randomly generated individuals, when most of the population is too similar to each other. Diversity is important in genetic algorithms (and genetic programming) because crossing over a homogeneous population does not yield new solutions. In evolution strategies and evolutionary programming, diversity is not essential because of a greater reliance on mutation.

Operating on dynamic data sets is difficult, as genomes begin to converge early on towards solutions which may no longer be valid for later data. Several methods have been proposed to remedy this by increasing genetic diversity somehow and preventing early convergence, either by increasing the probability of mutation when the solution quality drops (called *triggered hyper mutation*), or by occasionally introducing entirely new, randomly generated elements into the gene pool (called *random immigrants*). Again, evolution strategies and evolutionary programming can be implemented with a so-called "comma strategy" in which parents are not maintained and new parents are selected only from offspring. This can be more effective on dynamic problems.

GAs cannot effectively solve problems in which the only fitness measure is a single right/wrong measure (like decision problems), as there is no way to converge on the solution (no hill to climb). In these cases, a random search may find a solution as quickly as a GA. However, if the situation allows the success/failure trial to be repeated giving (possibly) different results, then the ratio of successes to failures provides a suitable fitness measure.

For specific optimization problems and problem instances, other optimization algorithms may be more efficient than genetic algorithms in terms of speed of convergence. Alternative and complementary algorithms include evolution strategies, evolutionary programming, simulated annealing, Gaussian adaptation, hill climbing, and swarm intelligence (e.g.: ant colony optimization, particle swarm optimization) and methods based on integer linear programming. The suitability of genetic algorithms is dependent on the amount of knowledge of the problem; well-known problems often have better, more specialized approaches.

Variants

Chromosome representation

The simplest algorithm represents each chromosome as a bit string. Typically, numeric parameters can be represented by integers, though it is possible to use floating point representations. The floating point representation is natural to evolution strategies and evolutionary programming. The notion of real-valued genetic algorithms has been offered but is really a misnomer because it does not really represent the building block theory that was proposed by John Henry Holland in the 1970s. This theory is not without support though, based on theoretical and experimental results (see below). The basic algorithm performs crossover and mutation at the bit level. Other variants treat the chromosome as a list of numbers which are indexes into an instruction table, nodes in a linked list, hashes, objects, or any other imaginable data structure. Crossover and mutation are performed so as to respect data element boundaries. For most data types, specific variation operators can be designed. Different chromosomal data types seem to work better or worse for different specific problem domains.

When bit-string representations of integers are used, Gray coding is often employed. In this way, small changes in the integer can be readily affected through mutations or crossovers. This has been found to help prevent premature convergence at so called *Hamming walls*, in which too many simultaneous mutations (or crossover events) must occur in order to change the chromosome to a better solution.

Other approaches involve using arrays of real-valued numbers instead of bit strings to represent chromosomes.

Results from the theory of schemata suggest that in general the smaller the alphabet, the better the performance, but it was initially surprising to researchers that good results were obtained from using real-valued chromosomes. This was explained as the set of real values in a finite population of chromosomes as forming a *virtual alphabet* (when selection and recombination are dominant) with a much lower cardinality than would be expected from a floating point representation.

Elitism

A practical variant of the general process of constructing a new population is to allow the best organism(s) from the current generation to carry over to the next, unaltered. This strategy is known as *elitist selection* and guarantees that the solution quality obtained by the GA will not decrease from one generation to the next.

Parallel implementations

Parallel implementations of genetic algorithms come in two flavors. Coarse-grained parallel genetic algorithms assume a population on each of the computer nodes and migration of individuals among the nodes. Fine-grained parallel genetic algorithms assume an individual on each processor node which acts with neighboring individuals for selection and reproduction. Other variants, like genetic algorithms for online optimization problems, introduce time-dependence or noise in the fitness function.

Adaptive GAs

Genetic algorithms with adaptive parameters (adaptive genetic algorithms, AGAs) is another significant and promising variant of genetic algorithms. The probabilities of crossover (pc) and mutation (pm) greatly determine the degree of solution accuracy and the convergence speed that genetic algorithms can obtain. Instead of using fixed values of *pc* and *pm*, AGAs utilize the population information in each generation and adaptively adjust the *pc* and *pm* in order to maintain the population diversity as well as to sustain the convergence capacity. In AGA (adaptive genetic algorithm), the adjustment of *pc* and *pm* depends on the fitness values of the solutions. In *CAGA* (clustering-based adaptive genetic algorithm),[13] through the use of clustering analysis to judge the optimization states of the population, the adjustment of *pc* and *pm* depends on these optimization states. It can be quite effective to combine GA with other optimization methods. GA tends to be quite good at finding generally good global solutions, but quite inefficient at finding the last few mutations to find the absolute optimum. Other techniques (such as simple hill climbing) are quite efficient at finding absolute optimum in a limited region. Alternating GA and hill climbing can improve the efficiency of GA while overcoming the lack of robustness of hill climbing.

This means that the rules of genetic variation may have a different meaning in the natural case. For instance – provided that steps are stored in consecutive order – crossing over may sum a number of steps from maternal DNA adding a number of steps from paternal DNA and so on. This is like adding vectors that more probably may follow a ridge in the phenotypic landscape. Thus, the efficiency of the process may be increased by many orders of magnitude. Moreover, the inversion operator has the opportunity to place steps in consecutive order or any other suitable order in favor of survival or efficiency. (See for instance or example in travelling salesman problem, in particular the use of an edge recombination operator.)

A variation, where the population as a whole is evolved rather than its individual members, is known as gene pool recombination.

A number of variations have been developed to attempt to improve performance of GAs on problems with a high degree of fitness epistasis, i.e. where the fitness of a solution consists of interacting subsets of its variables. Such algorithms aim to learn (before exploiting) these beneficial phenotypic interactions. As such, they are aligned with the Building Block Hypothesis in adaptively reducing disruptive recombination. Prominent examples of this approach include the mGA, GEMGA and LLGA.

Problem domains

Problems which appear to be particularly appropriate for solution by genetic algorithms include timetabling and scheduling problems, and many scheduling software packages are based on GAs [*citation needed*]. GAs has also been applied to engineering. Genetic algorithms are

often applied as an approach to solve global optimization problems.

As a general rule of thumb genetic algorithms might be useful in problem domains that have a complex fitness landscape as mixing, i.e., mutation in combination with crossover, is designed to move the population away from local optima that a traditional hill climbing algorithm might get stuck in. Observe that commonly used crossover operators cannot change any uniform population. Mutation alone can provide periodicity of the overall genetic algorithm process (seen as a Markov chain).

Examples of problems solved by genetic algorithms include: mirrors designed to funnel sunlight to a solar collector, antennae designed to pick up radio signals in space, and walking methods for computer figures.

In his Algorithm Design Manual, Skiena advises against genetic algorithms for any task:

It is quite unnatural to model applications in terms of genetic operators like mutation and crossover on bit strings. The pseudo-biology adds another level of complexity between you and your problem. Second, genetic algorithms take a very long time on nontrivial problems. [...] [T]he analogy with evolution—where significant progress require millions of years—can be quite appropriate.

I have never encountered any problem where genetic algorithms seemed to me the right way to attack it. Further, I have never seen any computational results reported using genetic algorithms that have favorably impressed me. Stick to simulated annealing for your heuristic search voodoo needs.

III. RESULTS AND DISCUSSION

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 * 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, are chosen.

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.

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.

IV. 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 location 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.

V. REFERENCES

[1] W. Zhu, C. Luo, J. Wang, and S. Li, "Multimedia cloud computing:An emerging technology for providing multimedia services and applications," 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] R.Yavatkar, D.Pendarakis, and R. Guerin, "A framework for policy based admission control," Internet Requests for Comments, RFC Editor, RFC 2753, 2000.

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

[7] 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.

[8] J.Sun, X.Wu, and X.Sha, "Load balancing algorithm with multiservice 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.

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

[10] 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.

[11] 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.

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

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

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

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