Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol 3, No.4, 2012

An Evolutionary Approach to Optimizing Cloud Services

Dipanshu Agrawal* Heera Lal Jaiswal Ishan Singh K. Chandrasekaran

Department of Computer Science and Engineering. National Institute of Technology, Surathkal, Karnataka, India – 575025

* E-mail of the corresponding author: dipanshu.agrawal@hotmail.com

Abstract

In this paper, we propose an optimized scheduling algorithm for cloud services. We propose a Genetic Algorithm for optimum allocation of Virtual Machines (VMs) that permit maximum usage of physical resources. We describe the fitness function and the GA operators in detail and how they manipulate the problem space.

Keywords: cloud computing, service optimization, genetic algorithm

1. Introduction

CLOUD COMPUTING is becoming more and more popular in organizations around the world as it allows sharing of computing resources that are distributed all over the world. It is generally used for on-demand storage and processing power. The user/software agent accesses computing resources as general utilities that can be leased and released. The main benefits are the user avoids up-front costs, lower operating costs, reduced maintenance cost, and scalability on demand. The cloud features adaptability to user's need.

In Cloud computing, each business process is broken down into a set of abstract services. Each service encapsulates some functionality using its interface. A concrete service is selected at runtime to fulfill the function. A Service Level Agreement (SLA) is defined upon a business process as its end-to-end Quality of Service (QoS) constraints. Different concrete services operate at different standards of QoS, and an appropriate set of concrete services needs to be selected so that it guarantees the fulfillment of SLA, while keeping the costs as low as possible. This problem is a combinatorial optimization problem which is known to be NP-hard i.e. the number of solutions is huge, and there is no efficient algorithm to search for the optimal solution.

Genetic Algorithms have been known to produce near-optimal solutions to NP-hard problems efficiently (Davis, 1991). It is a search heuristic that mimics the process of natural evolution. This 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.

Problems which appear to be particularly appropriate for solution by genetic algorithms include timetabling, optimization and scheduling problems, and many scheduling software packages are based on GAs. GAs have also been applied to engineering. Many of their solutions have been highly effective, unlike anything a human engineer would have produced, and inscrutable as to how they arrived at that solution. (Goldberg D. E., 1989)

Hence we propose that VM service allocation problem be attempted to solve using a Genetic Algorithm Framework.

2. Related Works

Many Approaches to scheduling cloud services were presented in some papers (A. K. Amoura, 2002) (M.A. Iverson, 1999) (G. Koole, 2008). Most of these algorithms do not consider the interdependencies between the services and the communication constraints involved between them. The majority of complex cloud applications invoke multiple services which often communicate with each other. This kind of collaboration must be taken into account while allocating resources.

The Dependence allocation problem of services is known to be NP-Hard (G. Christodoulou, 2007) (Hochbaum, 1996) (K. Lenstra, 1990). Many Heuristic algorithms are present to solve this problem to a near-optimal solution. However there is a lack of practicable solution for cloud computing systems because most as such are multiple-QoS constrained.

The existing algorithms present in Open-Nebula and Eucalyptus [9] are constrained heavily, as they use very simple algorithms that may though provide an acceptable solution, but generally fail to produce the optimal solution. Eucalyptus offers a Greedy algorithm and a Round-Robin algorithm. While the Greedy algorithm looks at the immediate future need, the Round-Robin algorithm just cycles between available Virtual machines to allocate resources to them. (Horgan, 2011)

Open Nebula [10] on the other hand uses slightly more sophisticated algorithms. Its uses match-making i.e. for each immediate request it ranks all available VMs and selects the most highly ranked. Ranking policies are aimed at minimizing the number of nodes in use or maximize the resources available to each VM in a node. (Thiago Damasceno Cordeiro, 2010)

3. Problem Modelling

Consider a business process containing n tasks T1, T2, T3⋯..Tn. For each task Ti ($1 \leqslant i \leqslant n$) there are li candidate services that can perform the task. Users may impose some constraints on the amount of resources consumed such as execution time, price etc. Let there be m resources (r1, r2, r3……rm) that are constrained. Then, the QoS-based service selection problem involved in service composition is, in fact, how to select one service for each involving task from its corresponding existing candidate service group, so that the overall QoS of the constructed composite service can be maximized while the constraints set by users are satisfied. These services can be modeled as Directed Acyclic Graph, an example of which is shown in figure 1.

The arc forms the dependency set dij, where Ti is called the parent task of Tj, and dij is the data produced by Ti and consumed by Tj. For each task Ti ($1 \leqslant i \leqslant n$) there is a set of services that can fulfill that task. These are called the candidate service instances si = (si1, si2, si3…smi)...

We need to form the optimum selection of candidate services such that the overall QoS is maximized.

Let fij be the QoS score of the jth candidate service for the ith task. It can be formulated as:

$$Max\left(\sum_{i-1}^{n}\sum_{j-1}^{l_i} x_{ij} f_{ij}\right) \quad (1)$$

$$\sum_{i-1}^{n}\sum_{j-1}^{li} x_{ij} r_{ij}^{k} \leq R^{k} \quad (2)$$

Where xij = 1 if Task ‘i’ is fulfilled using service ‘j’otherwise 0. And R is the upper limit of the resource r set by the user.

4. Optimization Policy

We would not get into details of any network topologies and assume the cloud to be Centralized.

Let us consider a set of VM requests, a set of interconnected computing nodes connected by a network. The computing nodes are different kinds of ordinary PCs, servers, and even high performance clusters. The cloud provides virtual machines that run on these physical resources to the client which accesses them through internet. The CPU speed (usually the number of cores), Memory capacity and Hard Drive capacity in consideration, which is most of the existing IaaS cloud systems do.

Eucalyptus uses Greedy (First fit) and Round robin scheduling strategies. Greedy query all the computational resources from the first to the last node until finding a suitable node every time new request comes and deal with them one by one for multiple requests. Round robin records the last position of the scheduler visited. And the scheduler starts from the last visited position next time new request(s) come(s) meanwhile the resources are considered as a circular linked list. Open Nebular uses Haizea, an open source VM-based lease management architecture as the scheduler and provides the queuing system, advanced reservation, preemption, immediate lease strategies, etc. All these policies pay more attention to "when" but neglect "how", the utilization of resources. Nimbus can be configured to use familiar schedulers like PBS (Portable Batch System) or SGE (Sun Grid Engine) to schedule virtual machines (V. Kolosov, 2004) (Gentzsch, 2001) (Yanggratoke, 2009). PBS is a queuing system and SGE uses Job Scheduling Hierarchically (JOSH), both do not have a good utilization of resources.

5. Conceptual Model

The figure 2 depicts how the optimization technique works. All the service requests a made through standard gateway, that uses the algorithm described in the following sections for optimum allocation.

4. Optimization Algorithm

In a genetic algorithm, a population of strings (called chromosomes or the genotype of the genome), which encode candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem, evolves toward better solutions. 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 happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population 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. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached (Banzhaf, Nordin, Keller, & Francone, 1998).

6.1 Chromosome Representation

A chromosome (also sometimes called a genome) is a set of parameters which define a proposed solution to the problem that the genetic algorithm is trying to solve. The chromosome is often represented as a simple string; although a wide variety of other data structures are also used (Yang, Wu, Lee, & Liang, 2008).

The chromosomes encode a scheduler. They consist of several <task, service> pairs. Each pair means a mapping that task is mapped onto a service .It also indicate that the position of each particle satisfies the precedence constraint between activities.

6.2 Constraints

As we mentioned above, we considering one computing node just launch one instance at a scheduling time. And one job will be allocated for one time. But it is not means one physical machine can only run on one VM. We will gather all the idle resources of each computing node, and each of them will be appearing at the scheduling time except that the node do not have any free resource to hand out. For example, a 4-core computing node that half of its CPUs is in occupying. We will take the rest part of its CPUs in considering at scheduling time.

6.3 Fitness Funciton

Each design solution is represented as a string of numbers (referred to as a chromosome). After each round

of testing, or simulation, the idea is to delete the 'n' worst design solutions, and to breed 'n' new ones from the best design solutions. Each design solution, therefore, needs to be awarded a figure of merit, to indicate how close it came to meeting the overall specification, and this is generated by applying the fitness function to the test, or simulation, results obtained from that solution. (Jin & Branke, 2005)

We define the fitness function using the total QoS score. However for each resource that is violating its user-set constraint, we impose a penalty$\propto$. This gives our combine fitness function as below.

$$\left(\sum_{i-1}^{n}\sum_{j-1}^{l_i} x_{ij} f_{ij}\right) - \propto \left(R^k - \sum_{i-1}^{n}\sum_{j-1}^{li} x_{ij} r_{ij}^k\right) \quad (3)$$

Where symbols have their usual meaning as defined in section 3.

6.4 Selection Algorithm

Selection is the stage of a genetic algorithm in which individual genomes are chosen from a population for later breeding (recombination or crossover). (Rennard, 2006)

The selection procedure we use is called the Roulette wheel selection. It is also known as fitness proportionate selection. The individual is selected on the basis of fitness. The probability of an individual to be selected increases as its fitness increases relative to its competitors.

A generic selection procedure may be implemented as follows:

1. The fitness function is evaluated for each individual, providing fitness values, which are then normalized. Normalization means dividing the fitness value of each individual by the sum of all fitness values, so that the sum of all resulting fitness values equals 1.
2. The population is sorted by descending fitness values.
3. Accumulated normalized fitness values are computed (the accumulated fitness value of an individual is the sum of its own fitness value plus the fitness values of all the previous individuals). The accumulated fitness of the last individual should be 1 (otherwise something went wrong in the normalization step).
4. A random number R between 0 and 1 is chosen.
5. The selected individual is the first one whose accumulated normalized value is greater than R.

If this procedure is repeated until there are enough selected individuals.

6.5 Cross-over Algorithm

In genetic algorithms, crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next. It is analogous to reproduction and biological crossover, upon which genetic algorithms are based. Cross over is a process of taking more than one parent solutions and producing a child solution from them. There are methods for selection of the chromosome (Gwiazda, 2006).

For the purpose of our problem we'll use the following cross over algorithm (Syswerda, 1989).

1. Let A be the first chromosome and B be the second chromosome.
2. Create two blank chromosomes C and D.

3. For every Task
    a. Select randomly from the <task, service> pair in A or B.
    b. Insert the selected pair in C.
    c. Insert the non-selected pair in D
4. C & D are the new chromosomes.

6.6 Mutation

In genetic algorithms of computing, mutation is a genetic operator used to maintain genetic diversity from one generation of a population of algorithm chromosomes to the next. It is analogous to biological mutation (Obitko, 2007). Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to better solution by using mutation. Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set low. If it is set to high, the search will turn into a primitive random search.

For mutation operator, we replace a random slice of chromosomes. The new random slice is generated by allocating services to task randomly from the pool of available candidate services.

1. Let Probability of mutation be p between 0 and 1.
2. Generate a random number n, uniformly distributed in 0 and 1
3. If(n < p)
    a. Let l be the no. of tasks.
    b. Generate a random number r1 with 1<=r1<l
    c. Generate a random number of r2 with r1<r2<=l
    d. For every task between r1 and r2

6.7 Mutation

The termination condition describes when to stop the algorithm. It is important as over-optimization is also an issue. The termination condition is define as

1. The target QoS as per the Service Level Agreements is met.
2. The constraint user set on the specified resources is satisfied.

6.8 Genetic Algorithm

Once we have the genetic representation and the fitness function defined, GA proceeds to initialize a population of solutions randomly, and then improve it through repetitive application of mutation, crossover, and inversion and selection operators (Goldberg D. E., 1989).

1. Choose the initial population of individuals. This pool can be randomly generated or selected on the basis of some heuristics.
2. Evaluate the fitness of each individual in that population.
3. Repeat on this generation until termination (time limit, sufficient fitness achieved, etc.):
    a. Apply the selection procedure to get a n pairs of individual, where n = m/2 and m is the total size of population.
    b. Create an initially blank new population p_{next}.

c. For each pair
    i. Breed new individuals through crossover and mutation operations to give birth to offspring
    ii. Add new individuals to pnext.
d. Evaluate the new population and assign fitness value to each individual as per the described fitness function
e. Replace least-fit population with new individual

7. Conclusion

In this paper, based on our research on various IT optimization techniques, we proposed an optimized scheduling algorithm for cloud services. We used Genetic Algorithm to design our optimization model.

We first went on to describe the problem in a mathematical way, as it's a first step to automatic optimization. The problem was reduced to the problem of maximizing a function, which enable us to use familiar mathematics to solve it. Then we described various genetic operators such as selection, cross-over, mutation etc. which are used to drive the Genetic algorithm forward.

The algorithm could be coded inside the load balancer or the scheduler module of a cloud environment. As such it continuously optimize the allocation of resources and make sure the Quality of Service offered is maximized.

References

A. K. Amoura, E. B. (2002). Scheduling Independent Multiprocessor Tasks. *Algorithmica, 32*, 247-261.

Banzhaf, W., Nordin, P., Keller, R., & Francone, F. (1998). *Genetic Programming – An Introduction.* San Fransisco, CA: Morgan Kaufmann.

Davis, L. (1991). *Handbook of Genetic Algorithm* . United States of America.

G. Christodoulou, E. K. (2007). A lower bound for scheduling mechanisms. In *SODA* (pp. 1163-1169).

G. Koole, R. R. (2008). Resource Allocation in Grid Computing. *Journal of Scheduling, 11*, 163-173.

Gentzsch, W. (2001). Sun Grid Engine: Towards Creating a Compute Power Grid. *Cluster Computing and the Grid - CCGRID* , (pp. 35-39).

Goldberg, D. E. (1989). *Genetic algorithm in search* .

Goldberg, D. E. (1989). *Genetic Algorithms in Search Optimization and Machine Learning.* Addison Wesley.

Gwiazda, T. D. (2006). *Genetic Algorithms Reference Vol.1 Crossover for single-objective numerical optimization problems.* Lomianki.

Haizea. (n.d.). Retrieved from http://haizea.cs.uchicago.edu.

Hochbaum, D. (1996). *Approximation algorithms for NP-hard problems.* Manhattan, United States of America: PWS Publishing Co. Boston.

Horgan, T. (2011, September 28). *Eucalyptus – Open Source Cloud Platform*. Retrieved from CIT Cork Ireland: http://cloud.cit.ie/2011/09/842/

Jin, Y., & Branke, J. (2005). Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation*.

K. Lenstra, D. S. (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming, 1*(46), 259-271.

M.A. Iverson, F. O. (1999). Statistical prediction of task execution times through analytic benchmarking for

scheduling in a heterogeneous environment. *IEEE Transactions on Computers*, *48*(12), pp. 1374-1379.

Obitko, M. (2007). *XI. Crossover and Mutation*. Retrieved from http://www.obitko.com/: http://www.obitko.com/tutorials/genetic-algorithms/crossover-mutation.php

Rennard, J.-P. (2006, May 6). *Introduction to Genetic Algorithms*. Retrieved from www.rennard.org: http://en.wikipedia.org/wiki/Selection_(genetic_algorithm)

Syswerda, G. (1989). Uniform Crossover in Genetic Algorithms. *International Conference on Genetic Algorithms - ICGA*.

Thiago Damasceno Cordeiro, D. B.-E. (2010). Open Source Cloud Computing Platforms . *Grid And Cooperative Computing*, (pp. 366-371).

V. Kolosov, Y. L. (2004). Monitoring system for OpenPBS environment. *Nuclear Instruments & Methods in Physics Research Section A-accelerators Spectrometers Detectors and Associated Equipment - NUCL INSTRUM METH PHYS RES A*, 135-137.

Yang, J., Wu, C., Lee, H., & Liang, Y. (2008). Solving traveling salesman problems using generalized chromosome genetic algorithm. *Progress in Natural Science*, 887-892.

Yanggratoke, R. (2009, April 27). The Amazon Web Services .

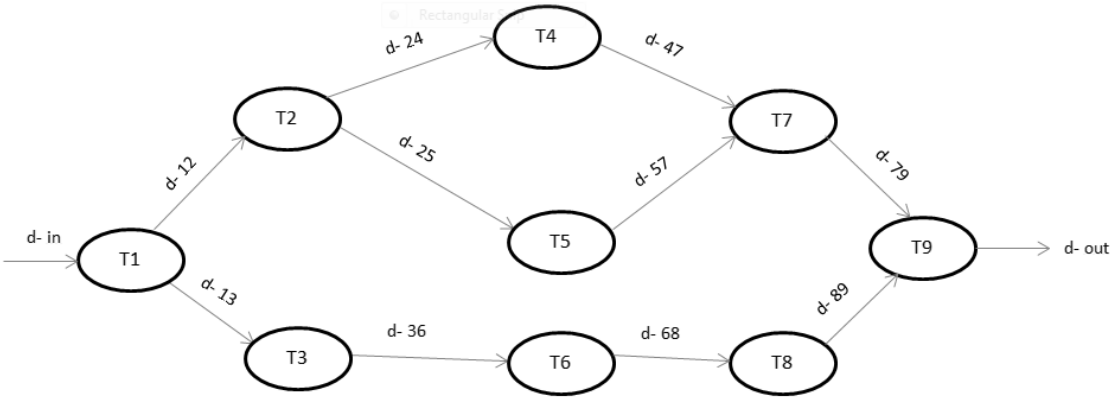

Figure 1. Process Representation

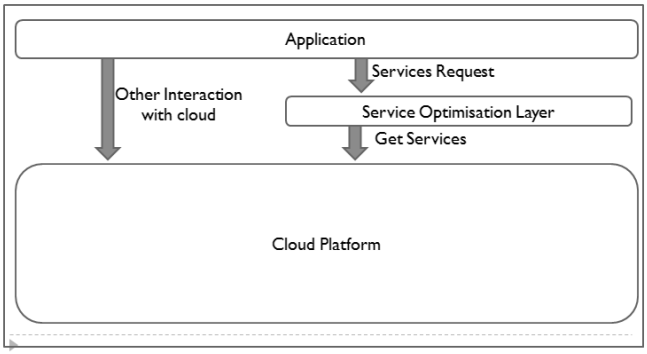

Figure 2. Conceptual Design