Independent Tasks Scheduling Based on Genetic Algorithm in Cloud Computing

Chenhong Zhao[1] hongchenzhao@163.com
Shanshan Zhang[1] lvzongping2006@163.com
Qingfeng Liu[1] sithlqf@gmail.com
Jian Xie[1] xiejian1990@gmail.com
Jicheng Hu[2] jicheng@whu.edu.cn

[1]International School of Software, Wuhan University
Wuhan 430079, China
[2]State Key Laboratory of Software Engineering, Wuhan University
Wuhan 430072, China

Abstract — **Task scheduling algorithm, which is an NP-completeness problem, plays a key role in cloud computing systems. In this paper, we propose an optimized algorithm based on genetic algorithm to schedule independent and divisible tasks adapting to different computation and memory requirements. We prompt the algorithm in heterogeneous systems, where resources (including CPUs) are of computational and communication heterogeneity. Dynamic scheduling is also in consideration. Though GA is designed to solve combinatorial optimization problem, it's inefficient for global optimization. So we conclude with further researches in optimized genetic algorithm.**

Keywords - Cloud computing; Dynamic scheduling; Genetic Algorithm; Heterogeneous System

I. INTRODUCTION

Task scheduling, one of the most famous combinatorial optimization problems, plays a key role to improve flexible and reliable systems. The main purpose is to schedule tasks to the adaptable resources in accordance with adaptable time, which involves finding out a proper sequence in which tasks can be executed under transaction logic constraints.

There have been a lot of algorithms put forward for task scheduling in many different research fields, whereas few covers the arisen field -- cloud computing.

Cloud computing, which means assigning computation on a dynamic resource pool composed of massive computers, makes users gain computation capability, memory space and software services online according to different requirements. In cloud computing, resources (including CPUs) are of computational and communication heterogeneity and are always dynamically collocated. This leads task scheduling in cloud computing to be a dynamic scheduling problem. There are mainly two factors of uncertainties [1]:

- Task flow is uncertain. Instances of task flow are uncertain; the execution path and times of the flow are also uncertain.
- Resources are uncertain. During the possible long time of execution, available resources with their quantity and form are changing all the way; resources' capability, current load, interests and tasks' requests, which can effect the scheduling a lot, are dynamic too.

Genetic algorithm, based on natural selection and inheritance theory, has been widely and successfully applied in scheduling problems.

In this paper, we introduce an application of GA in task scheduling in order to adapt to the memory constraints and high request of performance in cloud computing. The paper is organized as follows: the second part will be depicting dynamic scheduling and some relative researches in this field; the third part gives a brief representation of cloud computing; the forth part prompts our algorithm; the last part lays out our experiment results and concludes with a possible optimized algorithm.

II. DYNAMIC SCHEDULING

A. Introduction

Scheduling problem is aimed at resources distribution and utilization in a limited environment. It can be defined into two kinds: static scheduling and dynamic scheduling. Static scheduling uses so-called prescheduling technology to schedule known tasks in foregone environment, while dynamic scheduling must depend on not only the foregone tasks and environment but also the current system state to make scheduling plan[5~7].

B. Relative Rearches

Dynamic scheduling problem is an NP-complete problem, whose cost of precise computation will increase exponentially with the increase of the problem scale. In fact, we can just find out an approximate solution to NP problem [10].

At present, three general algorithms become popular to deal with such problems: Simulated Annealing; Tabu Search; Genetic Algorithm. Among them, GA gains wide attention because of its simpleness and parallel search.

III. CLOUD COMPUTING

In this paper computing resources in a cloud computing system is modeled to have a topological space which each node (computing resource) is assigned as a local resource for some other nodes by connecting weights. Mapping and scheduling of macro data-flow graphs to a series of suitable resources from the clouds is a fundamental and challenging problem. We will propose an architecture models to minimize the application execution time.

Sponsored by the National Innovative Project

978-1-4244-3693-4/09/$25.00 ©2009 IEEE

Looking through the lens of an architect, architecting software as an application in cloud computing system is an area sorely in need of better modeling. Cloud computing system delivers computation as a service. In the mean time SaaS delivery mechanism will become a main software business model and will change the economics of the software industry in a way that allows online information and computing to be accessible to many more people in emerging cooperation works such as India, China and US, by the technology of cloud computing.

Cloud computing system scales applications by maximizing concurrency and using computing resources more efficiently. One must optimize locking duration, statelessness, sharing pooled resources such as task threads and network connections bus, cache reference data and partition large databases for scaling services to a large number of users.

Our system model is based on SOA (Service Oriented Architecture). Each basic element of the system is considered to be a service node. Every service has a hierarchical hypergraph structure. Under this model we have two kinds of basic procedures: EP (Enwrapping Procedure) and SP (Solution Procedure). The EP phase is to enwrap computing resource into cloud computing system in which a computing resource is semantically parsed and enwrapped as a node service. The SP phase is to discretize the input user's computational problems to be a series of services through a semantic translator. This series is sent to a solver to be solved and the result is sent back to semantic translator and the composed to be a solution to the input user problem.

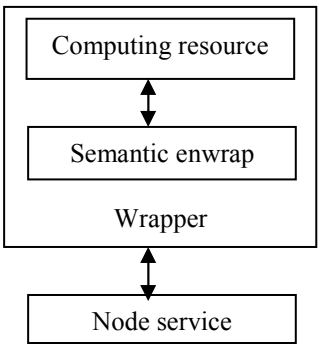

Figure 1. Enwrapping Procedure

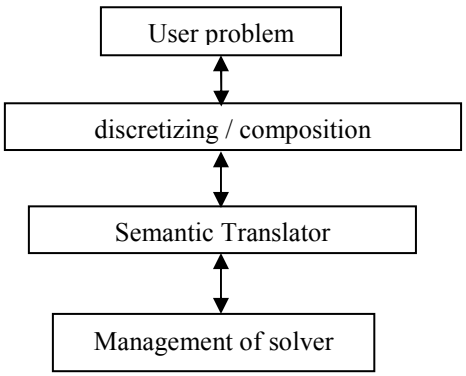

Figure 2. Solution Procedure

IV. SOLUTION BASED ON GA

A. Fundamental Concepts

1) Task Model

Suppose that a cloud computing system consist of heterogeneous process unit possess of $\boldsymbol{m}$($\boldsymbol{m}$>1) units. The tasks in this system features are as follows:

a) Tasks are aperiodic; i.e., task arrival times are not known a priori. Every task has the attributes arrival time (a_i), worst case computation time (c_i), and deadline (d_i). The ready time of a task is equal to its arrival time.

b) Tasks are nonpreemptive; each of them is independent.

c) Each task has two ways of access to a process unit:(1) exclusive access, in which case no other task can use the resource with it, or (2) shared access, in which case it can share the resource with another task (the other task also should be willing to share the resource) [2].

2) Scheduler Model

As with [3,4] in references, this paper also assume centralized scheduling scheme; i.e., a master processor unit in cloud, collecting all tasks, will take charge of dispatching them to other process units. Each process unit has its own dispatch queue (DQ). The master unit communicates with other process units through these dispatch queues. This organization ensures that processor units always find some tasks in the dispatch queue when the finish the execution of their current task. The master unit works in parallel with other units, scheduling the newly arrived tasks, and periodically updating the dispatch queues. Tasks are sorted ascending by the value of deadlines.

B. Genetic Algorithm Scheduling

1) Primary Match

Definition 1: CRV (Capability Requirement Vector) is a m-component vector for every task. $CRV_j(T_i)$ is the jth value of the capability task i requires. For example, CRV_1can be the arrival time of task *1*

Definition 2: CSV (Capability Supply Vector) is a m-component vector for every process unit. $CSV(P_j)$ is the jth value of the capability process unit i supplies. For example, CSV_1can be the available time of process unit *1*.

We now use the two vectors to get possible solutions.

In this phase, we suppose any process unit is available to each task. Then the primary match algorithm will be:

For task *i*
 For processor unit *j*
 If $CRV_j(T_i) \subset CSV(P_j)$
 Put P_j into Candidate Queue of T_i
 Else
 Combine with next CSV
 If $CRV_j(T_i) \subset$ CombinedCSV1
 Put $\{P_j, P_{j+1}\}$ into Candidate Queue of T_i
 Else
 Combine with next CSV
 If $CRV_j(T_i) \subset$ CombinedCSV2
 Put $\{P_j, P_{j+1}, P_{j+2}\}$ into CQ of T_i
 End If

End IF
End IF
End For
End For

Comments:

a) Candidate Queue (*i*)is for task *i* from which task *i* can choose a process unit to be executed.

b) The combination between CSVs is simply a mathematic addition of the vectors multiplying a coefficient matrix.

c) We consider at most three combinations for one task taking the cost of cooperation and communication with more combination into account.

2) Schedule Algorithm

Genetic algorithm is a new technology in the artificial intelligence, recently developed to the very popular inter-discipline .Genetic algorithms and genetic rules are in essence a search optimization technology, which starts from one initial population, in compliance with the "survival of the fittest survival "principle, through operators such as reproduction, hybridization, transformation, and forced to run continuously and iterate to approximate to the optimal solution[8]. Solution to the problem of the concrete steps are as follows:

a) encoding:

Coding program is to perform optimizing the selection of the solution of a problem by a way of code, so that it corresponds the problem solution space and genetic algorithm code space.

Suppose tasks including (TA_1, TA_2, ⁄, TA_n), for $\forall TA_i$, exists a total of N process unit or combination of process units to meet the conditions of its execution. Encoding with digital string, as shown in Figure 3.

TA1	TA2	• • •	TAi	• • •	TAn
P1	P2	• • •	Pi	• • •	Xn

Figure 3. Encoding with digital string

For TA_i, there is a selection of a total of N process units available. Each task can only select one process unit, so P_i represent that task TA_i will be executed by the *i*th of provided N process units up for selection ,and $1 \le i \le N$. At the same time, introducing the conflict measurement and counteract results to the code to ensure that each chromosome in the feasible solution space.

b) the initial population

In this paper, the algorithm generates initial population as follows:

1) initialization: set the size of population POP as *popsize*.

2) begin execution: the initial number of chromosome in the population is *ChromNum = 0*.

3) IF *ChromNum= popsize*, end algorithm, output the initial population, otherwise execute following steps.

4) Chromosome number for *ChromNum = ChromNum +1*, the *ChromNum*th chromosome in the population is *POP[ChromNum*], and initialize the gene number of the chromosome ,*geneNum = 0*.

5) IF *geneNum = n*, turn to step 3), otherwise the implementation of the following steps;

6)set the number of gene of the chromosomes for *geneNum = geneNum +1*, to obtain $TA_{geneNum}$ available process units number $n_{geneNum}$, and then randomly generate a integer between 1 to the integer $n_{geneNum}$ as the location of the gene; duplicate 5).

c) fitness function

Dynamic JSS problem must meet the deadline, so this paper assumes if a task fails to meet the deadline, it will get a punishment for its delay.

Objective function:

$$F = \min\{\max_{k\in(1,2,3,\ldots,m)}\{c_k\} + \sum_{i=1}^{n} f_1(d_i - T_i)\}$$

C_k is the finish time of the *k*th process unit. Punish function $f_1(x) = \begin{cases} -W_L X, X<0 \\ WeX, X\ge 0 \end{cases}$, d_i is the deadline of task *i* while T_i denotes the actual finish time of task *i*. W_L regulates the cost of delay and W_e regulates the cost of store for advanced tasks.

From above , *F* is an overall function for all tasks

d) selection operator

Here we choose a competitive mechanism based on local selection of the method .

Stage1 Randomly select from among a certain number of individuals (known as the marriage of scale),

Stage2 Compare the relative fitness value as selection criteria, and choose the high fitness ones into the next generation. General scale selection for marriage 2, and the numerical value of the fitness value is not directly proportional so this avoids the effects of super-individual, to a certain extent, avoids premature convergence and stagnation.

e) crossover operator

Suppose the crossover rate for bad individual is C_1 , and for good one is C_2 . The formula to compute *C* is:

$$C = \begin{cases} f * C_1, f \le \bar{f} \\ C_1 - \dfrac{(C_1 - C_2)*(f - \bar{f})}{f_{\max} - \bar{f}}, f > \bar{f} \end{cases} \quad (1)$$

The procedure of crossover is:

1) Get two chromosomes *POP[i], POP[j]* from previous generation and work out *CFitness[i], CFitness[j]*.

2) Get the crossover ratio by (1) for the two chromosomes.

3) Generate a random digit *c=Random(0,1);*

If *C<c,* generate an assembly as D composed of gene points with difference, set $D=\{d_1, d_2, \wedge, d_m\}$.

If D is empty, leave the chromosomes to the next generation directly.

Else, randomly crossover two gene points with difference, and hold the chromosome with higher fitness value in the next generation in order to reduce degradation.

f) mutation algorithm

Suppose a formula for self-adaptive mutation rate *M*:

$$M = \begin{cases} f * M_1, f \le \bar{f} \\ M_1 - \dfrac{(M_1 - M_2)*(f - \bar{f})}{f_{\max} - \bar{f}}, f > \bar{f} \end{cases} \quad (2)$$

The procedure of mutation is:

1) Work out the mutation ratio for every chromosome according to formula (2).

2) Select a chromosome randomly using the selection algorithm mentioned above, set it as *POP[i];*

3) At the mutation position *i,* generate a new random digit for it and save as *POP[new];*

4) Compute *CFitness[new];*

5) IF *CFitness[new]* ≥ *CFitness[i]*, *POP[new]* is added into the population, else, save the previous chromosome

g) termination algorithm

In this paper, the total selection process will be able to copy the chromosome that has the highest fit in the population .The termination of genetic algorithm is to reserve the chromosome that 20 generation consecutively keeps the highest fit value, when the conditions meet the termination will be based on fitness value of the size of sorted chromosomes, output the results to the decision makers to make decisions.

V. EXPERIMENT AND FUTURE WORK

Our experiment is designed to across different platforms where two tasks with distinct CRVS to be dispatched on two process units, one of which is 32-bit while the other is 64-bit. The two process units can be presumed as mini cloud computing system.

The primary matches were first collocated, we continued our experiment by setting population size per generation: 20, crossover rate for bad individual: 0.9, crossover rate for better ones: 0.4, the largest crossover rate: 0.2, the least crossover rate: 0.1, W_L: 1.0, W_e: 1.0. The algorithm ended at the 85th generation. Experiment sums up as follows:

1) The optimal fitness value at the 100th generation reached 2.9.
2) The utilization of resources is highly developed.
3) The ration between expected execution time and react execution time is 15/8.

The algorithm in this paper synthetically takes both time utilization and resource utilization into consideration, so the result is of high signification.

Researches of task scheduling in cloud computing tends to be more and more popular in the future, because the more efficient algorithm is used, the more benefits we can get. Traditional ways of Job-shop scheduling often leads to over-costed problem in cloud computing, while genetic algorithms can gain overall scheduling optimization.

In the future, more attention will be paid on reducing the solution space in GA, which is of especial importance in cloud computing for its integration of innumerable resources and tasks.

REFERENCES

[1] XIAO Zhi-Jiao, CHANG Hui-You,YI Yang “An Optimization M ethod of W orkflow Dynamic Scheduling Based on Heuristic GA ”, Computer Science, Vo1.34 No.2 2007.

[2] Indranil Gupta , G. Manimaran , and C. Siva Ram Murthy "A New Strategy for Improving the Effectiveness of Resource Reclaiming Algorithms in Multiprocessor Real-Time Systems", Journal of Parallel and Distributed Computing 60, 113133 (2000).

[3] Ramamritham K．, Stankovic A．J．. "Efficient scheduling algorithms for real time multiprocessor systems． IEEE Transactions on Parallel and Distributed Systems" 1990， 1(2)：184～194

[4] Qiao Ying．Wang Hong An， Dai Guo Zhong. "Developing a new dynamic scheduling algorithm for real-time multiprocessor systems"． Journal of Software， 2002， 3(1):5l～ 58(in Chinese).

[5] LeeCY，Piramuthu S.Tsai YK．"Job shop scheduling with a genetic algorithm and machine 1earning " Inr J. Pred Res.1997．35(4)：1171--1191．

[6] Wang L C, Chen H M, Liu C M．"Intelligent scheduling of FMS with inductive learning capability using neural networks"[J] lnt．J FMS 1995.7:147 一 l75．

[7] Nakasuka S ,Yoshida T "Dynamic scheduling system utilizing machine learning as a knowledge acquisition tool " lnt. J Prod Res. 1992，30 (2): 411—431

[8] L Ozdamar．A genetic algorithm approach to a general category project scheduling problem[J]．IEEE Trana Syst Man Cyber，1999，
29(1)：44—59．

[9] Saswati Sarkar “ Optimum Scheduling and Memory Management inInput Queued Switches with Finite Buffer Space”, 2003 IEEE 1373

[10] Radoslaw Szymanek and Krzysztof Kuchcinski, “ Task Assignment and Scheduling under Memory Constraints ” , 1089-6503/00$ 10.00 0 2000 IEEE