Design and Development of Agent Based Architecture for Effective Resource Utilization in a Grid Environment

P. DEEPAN BABU¹ and T. AMUDHA²

¹Department of IT & CT, VLB Janakiammal College of Arts and Science, Coimbatore, Tamil Nadu, (India).
²Department of Computer Applications, Bharathiar University, Coimbatore, Tamil Nadu, (India).
Corresponding author: pdeepan_13@yahoo.co.in, amudhaswamynathan@buc.edu.in

(Received: June 01, 2013; Accepted: June 06, 2013)

ABSTRACT

Software agents are the autonomous, problem-solving computational entities capable of effective operation in dynamic and open environments. Intelligent agent is a type of software agent, autonomous in nature which observes and acts upon environment and performs some task at each host. Agent can also coordinate, reason, solve a problem, clone and merge with other agents. A Grid is a set of resources (such as CPU, Memory, Disk, Applications, and Database) distributed over wide area networks and supports large scale distributed applications. Resources in grid are geographically distributed, linked through internet, to create virtual super computer for vast amount of computing capacity to solve complex problems. Genetic Algorithm works with key parameters such as fitness function, crossover probability and mutation probability and optimizes task scheduling. The paper proposes a software agent based architecture to utilize the resources effectively in Grid environment. The architecture is compared and analyzed the resource utilization with three algorithms namely Shortest job first, Arbitrarily Scheduling Algorithm and Genetic algorithm. The efficiency of resources utilization is analyzed and suitable algorithm is suggested.

Key words: Grid Computing, Software Agent, Evolutionary Technique, Genetic Algorithm, Load balancing.

INTRODUCTION

Grid Computing

Grid computing has emerged to satisfy high performance scientific computing with more computing power. Grid consists of geographically distributed computers linked through internet to create virtual super computer for collecting vast amount of computing capacity to solve complex problems. Foster and Kesselman¹ defined Grid computing as "A Computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities". Resources in grid are geographically distributed, linked through internet, to create virtual super computer for vast amount of computing capacity to solve complex problems.

Data grid primarily deals with data repositories, sharing, accessing and managing large amount of distributed data. A Data grid is a...
major type of grid\(^2\), used in data-intensive applications; where size of data files reach tera or sometimes peta bytes. High Energy Physics (HEP), Genetic and Earth Observation, are examples of such applications. Data Grid is an integrating architecture that connects a group of geographically distributed computers and storage resources that enable users to share data and resources.

Computational grid is developed to solve problems that require processing a large quantity of operations. Many research projects require lot of CPU time, some requires a lot of memory and some projects need the ability to communicate in real time. Today super computers are not enough to solve those needs. They don’t have the capacity, even if they did, it would not be economically justifiable to use these resources. Computational grids are the solution to all these problems and many more. Grid is a convenient way to connect many devices (e.g., processors, memory and IO-devices) so that end users are able to use all devices, computational powers combined for a certain amount of time.

Software Agents

The term agent derives from the present participle of Latin verb agree: to derive, lead, act or do. The American Heritage Dictionary defines an agent as “One that acts or has the power or authority to act or represent another or the means by which something is done or caused; instrument. Software agents are computer programs capable of flexible, autonomous action, the most complex form; agents may persist over time, capable of timely internal context dependent reaction to sensed events, plan and initiate unique series of actions to achieve stated goals and communicate with other agents or people toward those ends”.

A software agent is the software entity which functions continuously and autonomously in a particular environment often inhabited by other agents and process. The following are some features of software agents

- Reactivity, ability to selectively sense and act.
- Autonomy, goal directedness, proactive and self starting behavior.
- Collaborative, work in concert with other agents to achieve common goal.
- Knowledge level communication ability, ability to communicate with persons and other agents with more resembling human like speech acts.
- Inferential capability, act on abstract task specification using prior knowledge of general goals and preferred methods to achieve flexibility.
- Temporal continuity, persistence of identity and state over long period of time.
- Personality, capability of manifesting the attributes of a believable character such as emotion.
- Adaptivity, being able to learn and improve with experience.
- Mobility, being able to migrate in a self-directed way from one host platform to other

Evolutionary Algorithm

Evolutionary algorithms (EAs) are principally a stochastic search and optimization method based on the principles of natural biological evolution. Evolutionary Algorithms operate on a population of potential solutions, applying the principle of survival of the fittest\(^3\) to produce successively better approximations to a solution. At each generation of the EA, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and reproducing them using variation operators.

Genetic Algorithm

A Genetic algorithm (GA)\(^4\) 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.

A GA approach starts with a generation of individual. Individuals are encoded as strings known as chromosome and a chromosome corresponds to a solution to solve the above said problem. Each individual is evaluated by the fitness
function. Three major operations, selection, crossover and mutation operations are part of the GA based on some key parameters such as fitness function, crossover probability and mutation probability. All these parameters are used for the optimization of task scheduling.

Genetic algorithm was developed to simulate some of the processes observed in natural evolution, a process that operates on chromosomes. The following algorithm shows the general principle and working of Genetic algorithm.

**Algorithm GeneticAlgorithm ()**
1. Formulate initial population.
2. Randomly initialize population.
3. Repeat
   a. Evaluate objective function.
   b. Find Fitness function.
   c. Apply genetic operators.
      i. Selection.
      ii. Crossover.
      iii. Mutation.
   Until Condition Satisfies.

**Selection**

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. The method used to select the individual is Roulette-Wheel selection.

**Crossover**

A crossover operator is used to recombine two strings to get a better string. In the crossover operator new strings are created by exchanging information among strings of the mating pool. In One point crossover binary string from beginning of chromosome to the crossover point is copied from one parent, and the rest is copied from the second parent.

Fig 1 shows the Two point crossover, selects binary string from beginning of chromosome to the first crossover point from one parent, the part from the first to the second crossover point is from the second parent.

**Mutation**

Mutation is the next process after crossover, it alters one or more gene values in a chromosome from its initial state. The solution after mutation may change entirely from the previous solution. Hence, Genetic Algorithm may also give better solution by using mutation.

**Literature review**

The researchers Saeid Abrishami, Mahmoud Naghibzadeh and Dick Epema (2010) developed a model in which users negotiate with providers on required Quality of Service and on corresponding price to reach a Service Level Agreement. One of the most challenging problems in utility grid is scheduling, i.e., the problem of satisfying users’ QoS as well as minimizing the cost of workflow execution. They proposed a new QOS-based workflow scheduling algorithm based on a novel concept called Partial Critical Path. The algorithm recursively scheduled the critical path ending at a recently scheduled node. The algorithm minimized the cost of workflow execution while meeting a user-defined deadline.

Mustafizur Rahman, Rajiv Ranjan, Rajkumar Buuya and Boualem Benatallah (2011) have analyzed the Grid environment as the availability, performance, state of resources, applications, services, and data undergo continuous changes during the life cycle of an application. They stated that uncertainty is a major task in Grid environments by node failures, task dynamism, improper global knowledge, and finally heterogeneity of resources and job. To overcome the above challenges, the authors proposed comprehensive taxonomy that characterizes and classifies different software components and high-level methods that are required for autonomic management of applications in Grids. The
taxonomy not only highlights the similarities and differences of state-of-the-art technologies utilized in autonomic application management from the perspective of Grid computing, but also identifies the areas that require further research initiatives.

Ajith Abraham and Fatos Xhafa (2009) have written a survey paper on computational models for Grid scheduling problems and their resolution using heuristic and meta-heuristic approaches. Scheduling problems are the heart of Grid-like computational system. Scheduling based on different criteria, such as static versus dynamic environment, multi-objectivity, adaptivity and so on. The paper revealed the complexity of the scheduling problems in Computational Grids compared to scheduling in classical parallel and distributed systems, showed usefulness of heuristic and meta-heuristic approaches for the design of efficient Grid schedulers. The author’s discussed various requirements for a modular Grid scheduling and its integration with Grid architecture.

Jia Yu and Rajkumar Buyya (2005) have proposed a taxonomy that characterized and classified various approaches for building and executing workflows on Grids. The advent of Grid and application technologies, scientists and engineers are building more and more complex applications to manage and process large data sets, and execute scientific experiments on distributed resources. Such application scenarios require composing and executing complex workflows. Therefore, many efforts have been made towards the development of workflow management systems for Grid computing. The taxonomy not only highlights the design and engineering similarities and differences of up to date in Grid workflow systems, but also identifies the areas that call for further research such as Load balancing etc.

N. Sivakumar and K. Vivekanandan (2012) said that Agent technology is meant for developing complex distributed applications. Software agents are key building blocks of a Multi-Agent System (MAS). Software agents are unique in nature as it possesses certain distinctive properties such as Pro-activity, Reactivity, Social-ability, Mobility etc., Agent’s behavior might differ for same input at different cases and thus testing an agent and to evaluate the quality of an agent is a tedious task. The main objective of the authors is to come out with a set of measures to evaluate agent’s characteristics in particular reactive property. They applied five agents in online shopping system and evaluated, finally identified that the reactivity property and the results are encouraging.

Caddie Shijia Gao and Dongming Xu (2006) have formulated a conceptual model for Anti-Money Laundering using Simon’s decision-making process model. For a more adaptive, intelligent and flexible solution for anti-money laundering, the intelligent agent technology is applied in this research. Intelligent agents with their properties like autonomy, reactivity and proactivity are well suited for money laundering prevention controls. Several types of agents are proposed and open multi-agent architecture is presented for anti-money laundering. A prototype system for money laundering detection is developed to demonstrate the advances of the proposed system architecture with business values.

R. Tavakkoli-Moghaddam, N. ShamsavariPour, H. Mohammadi-Andargoli and M. H. Abolhasani Ashkezari, (2012) addressed the permutation of a flexible job shop problem that minimizes the makespan and total idleness as a bi-objective problem. They developed duplicate genetic algorithm (DGA) worked based on the GA, offers a better solution than the standard GA because it includes the rational and appropriate justification. It provides local search for the best solution in every generation with the neighborhood structure in several stages and stores them in an external list for reuse as a secondary population of the GA. The performance of the duplicate GA is evaluated by a number of numerical experiments. Comparing the results of the DGA with other algorithms realized that proposed DGA is efficient and appropriate for solving the given problem.

Xiaowei Yan, Chengqi Zhang and Shichao Zhang (2009) have designed a genetic algorithm based strategy for identifying association rules without specifying actual minimum support. The approach elaborate an encoding method and relative confidence is used as the fitness function.
With genetic algorithm, a global search performed and system automation is implemented, model does not require the user-specified threshold of minimum support. Furthermore, expand this strategy to cover quantitative association rule discovery. For efficiency a generalized FP-tree is implemented. Experiment evaluated and demonstrated algorithms have found significant reduction in computation costs.

Shen Wang, Bian Yang and Xiamu Niu (2010) have developed an extensive application of steganography, it is challenged by steganalysis. The most notable steganalysis algorithm is the RS attack which detects the steg-message by the statistic analysis of pixel values. To ensure the security against the RS analysis the authors presented a new steganography based on genetic algorithm. The LSB (least significant bit) of the message, the pixel values of the steg-image are modified by the genetic algorithm to keep their statistic characters. Thus, the existence of the secret message is hard to be detected by the RS analysis. Meanwhile, better visual quality achieved by the proposed algorithm. The experimental results demonstrated the effectiveness of proposed algorithm resistance to steganalysis with better visual quality.

Proposed architecture

Resource allocation is the major responsibility of Grid Environment where multiple requests arise from users with differing resource requirements. Figure 2 shows the proposed architecture, with layer approach. Each layer uses various Agent to deploy its service, the schedulers are operated with three techniques, Shortest Job first, Arbitrary Scheduling Algorithm, and Genetic Algorithm based scheduling.

User interface agent

Interface agents are computer programs that have ability to learn user preference and user habits and provide proactive and reactive assistance in order to increase the user’s productivity.

Job Queue and JDL

Job queue is a buffer in Resource Negotiator to hold user requested jobs for short time. Each request is heterogeneous in nature. To store User’s request efficiently Job Queue is used. JDL (Job Description Language) is essential in Grid based environment as Heterogeneous jobs are submitted. JDL is a specification that focuses on the description of computational task. JDL describes the aspects of job, Resource requirements to compute, and time taken for execution. The job is submitted to the Routing Agent, by verifying address from Global Addressing. Global Addressing is a mechanism used in environment to provide directory service.

Job Routing Agent

Learning and communication are major functionality performed by Job routing agent. Learning is a main advantage that allows the agents to initially operate in unknown environments and to become more competent than its initial knowledge. Job routing agent learns the request from Job Queue and identifies type of request through JDL. The Communicative Agent works only as an interface between two components and provides services to others. Communication is another function done by Job routing agent, it acts as an interface between Resource negotiator and Scheduler. The Main role of Communicative Agent is proper allocation of job to scheduler and after completion of task; it returns solution to User interface agents. Job routing agent maintains a queue called Job Dispatcher, a buffer region of memory storage used to temporarily hold data while it is being moved from one place to another.

Scheduler

Scheduler is a major component of this
The relation between the number of resources and the request definition are identified by user's application. User is responsible for deciding number of resources and how long they are needed. The request has two moldability parameter specification, amount of resources $R$ and time $T$.

Fig. 3 shows the moldable request, with four requests runs and four resources. The request and resources are specified by user. In this figure, Request-1 requires three resources to complete its task. After completion of work from Resource-1, Request-1 is allocated to Resource-2, after completion it is allocated to Resource-3. Moldable request uses two constraints which are operator precedence and machine constraints. Operator precedence constraint indicates that order of operations is not fixed and processing of operations cannot be interrupted. Machine constraint indicates single job should be executed at a time.

**Moldable job request**

The relation between the number of resources and the request definition are identified by user's application. User is responsible for deciding number of resources and how long they are needed. The request has two moldability parameter specification, amount of resources $R$ and time $T$.

Fig. 3 shows the moldable request, with four requests runs and four resources. The request and resources are specified by user. In this figure, Request-1 requires three resources to complete its task. After completion of work from Resource-1, Request-1 is allocated to Resource-2, after completion it is allocated to Resource-3. Moldable request uses two constraints which are operator precedence and machine constraints. Operator precedence constraint indicates that order of operations is not fixed and processing of operations cannot be interrupted. Machine constraint indicates single job should be executed at a time.

**Scheduling**

**Shortest Job First Scheduling Algorithm**

Shortest-job-first (SJF) scheduling algorithm associates with each process the length of the process's next processing time. When the resource is available, it assigns the process, which has smallest processing time next. The following algorithm shows the shortest job first approach.

**Algorithm Shortest Job First Scheduling(NumberOfNode)**

//NodeTime indicates time taken to complete its operation
1. Repeat For $I = 0$ to NumberOfNode by 1 do:
1. Repeat For J = I+1 to NumberOfNode by 1 do:
   a. If NodeTime[I] > NodeTime[J], then
      [Time Allocation.]
      i. Set Temp := NodeTime[I].
      [Node Allocation.]
   iv. Set Node2 := Node[I].
   [End of For.]
   [End of For.]
2. Call TurnATime( Node, NR, NM, Size)
3. Exit.

Arbitrary Scheduling Algorithm
This method uses the play of chance to assign a node by comparison groups in a trial, e.g. computer-generated random sequence. Arbitrary Scheduling implies that each individual or unit being entered into a trial has the same chance of receiving each of the possible interventions. It also implies that the probability an individual receive a particular intervention is independent of the probability that any other individual receive the same intervention.

Algorithm Arbitrary Scheduling
Algorithm Scheduling (NumberOfNode)
1. Set Count := 1, Z := 0.
2. Repeat While Z d' NumberOfNode, do:
   b. Set Z := Z+1. [End of While.]
3. Repeat While (Count>0), do:
   a. Set Y := 0.
   b. Set Alloval := MinusAllo(Alloval, NumberOfNode).
   c. Repeat For Z=0 to NumberOfNode, by 1 do:
      If(Alloval[Z]=-1), then
      Set Count:=Count+1.
      Else:
      Set Y := Y+1.
      [End of If.]
      [End of For.]
   d. If (Y = NumberOfNode), then
      Exit.[End of if.]
   e. If (Count>0), then
      1. Repeat For Z=0 to NumberOfNode, by 1 do:
         If(Alloval[Z]=-1), then
         [End of if.]
         [End of For.]
         4. Call TurnATime( Alloval, NR, NM, Size)
         5. Exit.
   f. If (A = 0), then
      i. Repeat For I = 0 to Limit by 1, do:
      i. If Node[i] = Sol, then
      1. Set Alloval[i] := Sol
      Else:
      Repeat For I = 0 to Limit by 1 do:
      If (Node[i] = Sol), then
      Set NodeCount := 1.
      [End of If.]
      [End of For.]
      [End of If.]
2. Return Array.

Genetic based Scheduling algorithm
Genetic based scheduling Algorithm Scheduling (NumberOfRequest)
1. Create InitialPopulation().
2. Set A := 0, Z := 0.
3. Repeat While Limit, do:
   b. Set NodeCount := 0, Count := 0.
   c. Call Selection().
   d. Call Crossover().
   e. Set Sol := Child.
   f. If (A = 0), then
      i. Repeat For I = 0 to Limit by 1, do:
      i. If Node[i] = Sol, then
      1. Set Alloval[i] := Sol
      Else:
      Repeat For I = 0 to Limit by 1 do:
      If (Node[i] = Sol), then
      Set NodeCount := 1.
      [End of If.]
      [End of For.]
      [End of If.]
2. Return Array.
A. If (NodeCount = 1), then
   Repeat For J = 0 to A by 1 do:
      If (Alloval[J] = Sol)
      Set Count := 1.
   [End of If.]
   [End of For.]
   [End of IF .]
B. If (Count = 0), then
Else:
   i. Call Mutation().
   ii. Set MNodeCount := 0, MCount := 0.
   iii. Set MSo1 := Child.
   iv. Repeat For X = 0 to Size by 1 do:
      If (Node[X] = MSo1), then
      Set MNodeCount := 1.
   [End of If.]
   [End of For.]
   v. If (MNodeCount = 1)
      i. Repeat For Y = 0 to A by 1 do:
         If (Alloval[Y] = MSo1), then
         Set MCount := 1.
      [End of if.]
      [End of For.]
   ii. If (MCount = 0), then
   [End of If.]
   [End of Step B IF.]
   [End of Step F IF.]
[End of Step 3 Loop.]
4. Exit

Algorithm Selection()
1. Set PRan1 := Random(Limit).
2. Set PRan1 := Random(Limit).

Algorithm CrossOver()
1. Set Fv := Random(Limit).
2. Set Sv := Random(Limit).
3. Repeat For I = 0 to Fv by 1 do:
   Set Child[I] := Parent1[I].
   [End of For.]
4. Repeat For I = Fv to Sv by 1 do:
   Set Child[I] := Parent2[I].
   [End of For.]
Table 1: Test Instances

<table>
<thead>
<tr>
<th>Author</th>
<th>Job Request</th>
<th>Instance Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Adams, E.</td>
<td>10</td>
<td>abz5, abz6</td>
</tr>
<tr>
<td>Balas and D. Zawack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H. Fisher, G.L.</td>
<td>10</td>
<td>ft10</td>
</tr>
<tr>
<td>Thompson</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S. Lawrence</td>
<td>10</td>
<td>la01, la02, la03, la04, la05, la16, la17, la18, la19, la20</td>
</tr>
<tr>
<td>D. Applegate, W. Cook</td>
<td>10</td>
<td>orb01, orb02, orb03, orb04, orb05, orb06, orb07, orb08, orb09, orb10</td>
</tr>
<tr>
<td>S. Lawrence</td>
<td>15</td>
<td>la06, la07, la08, la09, la10, la21, la22, la23, la24, la25</td>
</tr>
<tr>
<td>S. Lawrence</td>
<td>20</td>
<td>la11, la12, la13, la14, la15, la26, la27, la28, la29, la30</td>
</tr>
<tr>
<td>R. H. Storer, S. D. Wu, R. Vaccari</td>
<td>50</td>
<td>swv11, swv12, swv13, swv14, swv15, swv16, swv17, swv18, swv19, swv20</td>
</tr>
</tbody>
</table>

The second Layer is the Resource Negotiator. The Job Queue of Resource Negotiator takes an account of all the submitted jobs. The Job Description Language (JDL) describes the job specification. Then the Resource Negotiator assigns address for each job based on global addressing method and submits the job with address to a Job Routing Agent of Resource Negotiator. Third layer is the Local Scheduler which collects the respective job requests utilize...
collaborative agents and schedules the resources by communicating with the Fourth layer, Resource Layer. Resource Layer maintains resources from multiple locations.

The collaborative agents selects the resource and group them to form virtual organization with scheduler. The Scheduler applies the scheduling algorithms such as SJF, ASA, and GSA for scheduling jobs. Benchmark Problems are gathered to the test working efficiency of algorithm. The instances are taken from ORLIB. The architecture is implemented in Java with different Job request.

Table 1 shows various test instances tested in this architecture. The figures below show that Genetic based Scheduling algorithm have given best makespan as compared to Shortest Job First, Arbitrary Scheduling Algorithm. From the implementation of research work, it was observed that the Genetic based Scheduling Algorithm (GSA) is effective than Shortest Job First (SJF) and Arbitrary Scheduling Algorithm (ASA) in solving Moldable and Preemptive requests.

CONCLUSION

An Agent based Resource Negotiator has been proposed in this research work to allocate the resources in Grid Environment effectively based on user requirements. The Resource negotiator worked with three scheduling techniques, Shortest Job First (SJF), Arbitrary Scheduling Algorithm (ASA) and Genetic based Scheduling Algorithm (GSA). The performances of scheduling algorithms are tested with parameter resource utilization. The proposed architecture has proven to be highly effective in resource negotiation. Among the three algorithms implemented, Genetic based Scheduling algorithm (GSA) was identified best performing scheduling technique in the Grid architecture for effective resource utilization.

REFERENCES

Annual Conference of Asia Pacific Decision Sciences Institute, Hong Kong, (2006).


