IRACST - International Journal of Computer Science and Information Technology & Security (IJCSITS), ISSN: 2249-9555
Vol. 2, No. 1, 2012

ANZ based Task Allocation Algorithm (ANZA) for Computational Grid

Arun Baruah
133/4, 4th Crs, Munekolala
Marathalli, Bangalore
India
http://www.systemprogrammers.net/

***Abstract*—** Due to the advancement in the internet technologies, the processing power of the cpu and the low cost high speed bandwidth, a need arises for the geographically distributed network of computers to solve the problems, which requires high computing power and large number of resources. Thus, Grid computing was born. Grid Computing consists of geographically distributed and heterogeneous systems. They help to solve large-scale problems of science, engineering and commerce. There are several well known grids existing these days, which are BONIC, GLOBUS, CONDOR, ALCHEMI to name a few.
The main requirement and the challenge of the grid computing is the allocation of the tasks in the system. Since, its geographical distributed and heterogeneous in nature, it is very hard to come up with the most optimal resources for the given tasks to be executed. Many algorithms have been designed to solve this problem. We propose in this paper the use of Ant Colony Optimization based approach for task allocation (ANZA) and scheduling in computational grid. ANZA is massively distributed task allocation algorithm that takes the inspiration from biological ants, how the ants find their foods. The ant colony optimization technique is a population based search techniques for the solution of combinatorial optimization problem for resource discovery in grids. Making the use of pheromone trails (which evaporates in due course of time), the algorithms adapts effortlessly in the grid environment, which is prone to network failure. The use of distributed agents (ants) working in parallel and independent of each other for resource discovery anticipates the needs to maintain global state across the nodes. This help to save memory requirements. A detailed performance analysis is presented where we analyze the effect of various parameter settings of ANZA which helps to better understand the factors on which the resource allocation depends.

Keywords- Ant Colony Optimizations, Grid Computing, Task Allocation

Introduction

Grids consists of network of loosely coupled, geographically distributed computing resource, where end user submits a task to the grid and the Grid's resource manager (GRM) allocates the task to an appropriate grid nodes for execution. The aim is the find a resource that is capable of handling the task on hand, while considering the computation cost that arises as a result of using the resource measured in terms of grid $ and transmission network cost. The main aim is to minimize the cost and time. ANZA does the cost and time minimization in a massively distributed manner. Task allocation in grid is an NP hard problem. This paper proposes ANZA, a highly distributed algorithm for task allocation in computational grids.
Section I briefly explains other existing task allocation schemes. Section II provides an overview of ANZA. Section III describes the algorithms. Section IV describes the performance analysis of ANZA and Section V concludes this paper.

I. Existing Grid task allocation schemes:

A) Static Heterogeneous Energy Aware task Mapping: Heuristic static mapping [1], is ideal when nodes go down frequently, and the communication cost are very evident in comparisons to task processing cost, but scale poorly to large tasks.
B) Pooling: Pooling individual resources [2] to determine their applicability for a particular task, through pretty optimal for small grids, scales very poorly, and forces a long queuing time for tasks, irrespective of policy.
C) Ontological Task Definition and Allocation: This approach [3] uses user's input to determine the nature of the grid application and therefore determines mapping. This approach requires the availability of immediate comparisons between all resources, something not feasible is very large grid architectures.
D) Multiple Algorithm Spatial Modelling: This approach [4], very effective for mobile reconfiguration agent grids, requires massive computations to be carried out to determine the order of task allocation.

II. Overview of proposed algorithm ANZA

Ant Colony Optimization is a powerful technique often employed to solve optimization problems in a fixed search-space. It is computationally appealing as it is simple to implement and computationally robust with respect to local minima and maxima provided enough iterations are performed [5]. It also inherently parallel and can be implemented in a massively parallel way. ACO provides us with a biological metaphor of how decentralized systems of simple, interacting and often mobile agents can function collectively to yield complex behaviour. The emergent collective intelligent stems from the network of interaction that exist among individuals and their environment. We propose a ACO heuristic solution to the task allocation problem in Grid by programming the mathematical model of the behaviour of the ACO into mobile agents.
ANZA is implemented in a massively parallel manner and this contributes to its speed of allocation. The Global Resource Manager (GRM) which accepts the tasks from the user maintains two queues, one for the task that requires time optimization and the other requires cost optimization. As soon as the task is received by the GRM, it is queued into the appropriate queue depending on the scheduling policy specified by the user. Another module on the GRM is in charge of removing tasks from the queue and making them ready for allocation. We employ a Weighted Round Robin scheduling policy, where in the time optimization queue is emptied at faster rate then the cost optimization queue.
In order to test our algorithm, we have implemented a tired architecture where we have a LRM (local resource manager) below the GRM which hold administrative authority over a subset of Grid Resources that registered with it. The task that was removed from the queue stated above is handled over to all the LRM's which then individually compute the best allocation for this task making use of ANZA. From the LRM for each task, we deploy an Explorer Ants, which crawl the grid foraging for the best possible grid resource to allocate the task to. Each ant chooses its next hop on the basis of a

stochastic function that depends on two parameters. (i) The proximity of the grid resources to keep the communication cost and transmission time low. (ii) the trail intensity which is a function of the number of ants that have gone before on that link.
After running the algorithm for a specific number of cycles the emergent path appears leading to the heuristically best choice for allocating that particular task. Now that we have seen the context in which ANZA operates in the next session, we present the algorithms each of which running on different system that make up ANZA.

III. ANZA - Algorithms

ANZA algorithms have seven modules. They are: a) Scheduling algorithm, b) Task Handover algorithm, c) Ant-Deployment algorithm, d) Select next hop algorithm, e) Ant reception and evaluation algorithm, f) Allocator algorithm and g) De-allocator algorithm

a) Scheduling Algorithm:

```
Function AntScheduler(task t, policy p, cost c, time s)
Input:
  Task t
  Policy p -> where p is a scheduling policy which is either i) cost-optimized, ii) time-optimized, or iii)custom-optimized( parameters c & t specified by the end user)
Cost c, the grid$ threshold
Time t, the communication time threshold

Output:
  The queuing of the task partitions in the appropriate queue:
BEGIN
  if (p==cost-optimized)
     enque-cost-queue(t)
  else if (p== time-optimized)
     enqueue-time-queue(t)
  else if(s <= time-threshold-upper-limit && s>= time-threshold-lower-limit)
      enqueue-time-queue(t)
  else if(s <= cost-threshold-upper-limit && s >= cost-threshold-lower-limit)
      enqueue-cost-queue(t)
  end if
END
```

The scheduling module runs at the GRM and supports three types of scheduling policies, namely cost optimization, time optimization and custom parameter specification (CPS). For time optimization the objective is to minimize the time of execution to within a pre determined threshold and for cost optimization, the objective is to minimize the cost of execution, measured in Grid$ to within a similar threshold. After accepting the task, the task is queued into either the time queue or the cost queue as decided by the policy. In case of CPS the task is queued into either the cost queue or the time queue based on the value of the parameters specified.

b) Task handover Algorithm:

```
Function task-handover()
Input:
    none
Output:
    Dispatching task partitions to the appropriate β grid, where β grid is defined as the sub-grid under the administrative influence of an LRM.
BEGIN
   while time-weight times
      Current-task = dequeue-time-queue()
   for each LRM lr
      Send-to-lrm(lr, current-task)
   end while
   while cost-weight times
     Current-task = dequeue-cost-queue()
    for  each LRM lr
      Send-to-lrm(lr, current-task)
    end while
END
```

The task-handover module runs the GRM and selects the tasks from the two queues to send for allocation based on a weighted round robin scheduling policy. For each time frame, time-weight tasks are dequeued from the time queue and cost-weight partitions are dequeued from the cost queue. These tasks are handed over to the appropriate LRM to be allocated in its β Grid.

c) Ant-Deployment algorithm

```
Function Ant-Deployment(LRM lr)
Input:
LMR lr, the local resource manager that acts as the source for the deployments of ants.
Output:
   Ants are deployed onto the β grid under lr.
   grid-resource gr
   out-bound-vector obv
   ant-agent aa
while ant-maxcnt times
     aa = construct-ant()
     add-to-tabu-list(lg)
     gr = select-next-hop (obv, aa)
     aa.route-cost += link-cost
     aa.$cost += $cost(nexthop)
     transmit ant on choosen link
end while
```

The ants are deployed from the LRM to the corresponding β Grid. Each ant carries the task characteristics, the nodes visited so far, the route cost so far and the Grid$ spent so far. The ant also maintains a tabu list which contains a list of visited links. This list is maintained so that the ants can avoid travelled links and thus avoid cycles. The outbound vector maintains the link characteristics of all outbound links. The characteristics include, trail intensity of both cost pheromone and time pheromone and the communication cost as obtained by using a modified link state flooding approach. This link state flooding approach provides both neighbour identification and neighbour proximity detection. The ant selects the next hop stochastically based on the link characteristics of the neighbours which is contained in the out bound vector.

d) Select next hop algorithm

```
Function select-next-hop(out-bound-vector obv, ant-agent aa)
Input:
    Obv, an array of outbound links
Output:
     The choosen outbound link
BEGIN
    for ant aa, foreach entry in obv
```

```
  compute Pij for each entry in obv as,

  CTij(t) = pc  CTij(t)
  TTij(t) = pt TTij(t)
  if aa.Schedule-type ==cost-optimized
     pijkx(t) =[CTij(t)]α [C(i,j)]β/Nk
    Nk = ∑k in (S – Tabu(k)) [CTij(t)α[C(i, j)]β
  else if aa.Schedule-type == time-optimize
  pijkx (t) = [TTij(t)]α [C(i,j]β / Nk
  Nk = ∑k in (s – Tabu(k)) [TTij(t)]α [C(i, j)]β
Select return j with probability pij
END
```

This function selects the next hop given the out bound vector. The next hop is selected probabilistically based on the pheromone intensity and the cost of that link. The pheromone evaporation rate as specified by p makes the system responsive to dynamic network conditions. The fact that next hops are chosen stochastically and not deterministically ensures continuous exploration of alternative routes.

e) Ant reception and evaluation algorithm

```
Function ant-receipt-eval(ant-agent aa)
Input:
   ant-agent aa i.e., the ant received at this particular grid resource
Output:
   the received ant is either routed or sent back with success indication
   or sent back with failure indication
//variables
grid-resource gr
link ln //a linkto a neighbouring grid resource
if backtrack(aa)
   ln=nexthop(aa.path)
   if this is an ant that successfully found a grid resource for allocation,
      increase-trail-intensity(K)
   else if this is an ant that discovered a threshold violation
      decrease-trail-intensity(k)
   else if(ant.routecost >= time-threshold)
       gr=select-next-hop(obv,aa)
       add-to-tabu-list(gr)
       aa.routecost+=linkcost
       aa.$cost += $cost(nexthop)
       send ant on socket to grid resource gr
       return
  end if
   if(ant.route$ >=$_threshold)
      gr=select-next-hop(obv, aa)
      add-to-tabu-list(gr)
      aa.routecost+=linkcost
     aa.$cost +=$cost(nexthop)
      send ant via network link to grid resource gr
      return
   end if
   f=free-cycle-check()
   if(f >= ant.tack-size)
     optimality-index = µf + αMIPS_Rating
     Begin backtrack
     ln = nexthop(aa.path)
     send ant via network link to grid resource gr
  else
     gr=select-next-hop(obv,aa)
   end
```

An ant arriving at a Grid Resource can be either a back tracking ant or an ant that arrived here as an intermediate hop in its search for a potential grid resource for accommodating this task. A number of scenarios are possible. The ant of the latter case can find this resource suitable for allocation in which case it begins to backtrack using the path stored so far while rolling up pheromone levels proportionately based on the goodness of the fit. The goodness of the fit is measured by an optimality index that factors in the extent to which constraint satisfaction is achieved with respect to the scheduling parameters. It takes into account the resource characteristics. An ant can also abort its exploration from this node failing one or more of the thresholds in which case it begins to backtrack using the path stored so far while rolling down pheromone levels proportionately based on the goodness of the fit. If none of the above conditions are satisfied, the next hop is selected stochastically.

f) Ant reception and evaluation algorithm

```
Function send-allocator-ant()
Input:
  none
Output:
  an allocator ant is sent after convergence is achieved
for each ant-iteration
  receive all ants
  store each path and its count
  if p% of ants select the same path
    send allocator ant along same path
    for each edge ij in the path
      lower-trail-intensity()
      follow path ij
    end if
```

In the LRM, after a specified percentage of ants report back the same path, the allocator ant is sent to the chosen Grid resource to allocate memory and resources. As the allocator ant traces the path, it proportionately lowers the pheromone levels.

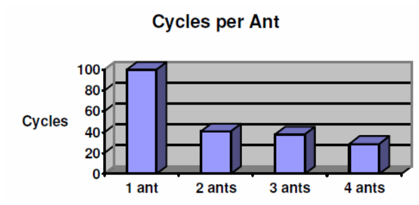
g) Deallocation algorithm

```
Function deallocate()
Input:
   none
Output:
   return deallocator ant
BEGIN
   wait-for-task-completion()
    free(memory)
    //using the path allocator ant used to reach the resource
    for each edge ij in the path
         lower-trail-intensity()
            follow path ij
END
```

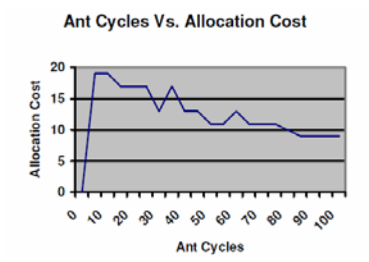
After the task completes its execution in the chosen grid resource, the memory and resources are freed. The deallocator ant which is actually the returning allocator ant returns to the LLGRB tracing the same path backwards, which appropriately increasing pheromone levels.

IV. Performance Analysis

The performance of ANZA depends on the parameter setting when it runs. Some of the parameters that need to be optimized are α (pheromone sensitivity index), β (cost sensitivity index) and p (pheromone evaporation rate). To arrive at the optimum parameter setting we had to rely on comprehensive experimentation on a trial and error basis.
In the first char given below, we have analyse the impact of the number of ants on the time taken to arrive at an optimum solution. For our sample problem, we present the results in the chart where it is evident that an increase in the number of ants per swarm results in faster convergence.



Next we studied the improvement in the allocation with the increase in the number of ant cycles. The results of this study are presented in the second chart below:



V. Conclusion

We have presented ANZA, a massively distributed algorithm that make use of ant optimization for allocating a task in a computational grid while trying to balance conflicting requirement of cost and time.

References

[1] Shivle et. al., "Static Mapping of Subtasks in Heterogeneous Ad Hoc Grid Environment", 13th Heterogeneous Computing Workshop (HCW) 2004
[2] Kurkovsky, S. and Bhagyavati, "Modelling a Computational Grid of Mobile Devices as a Multi-Agent System", Proceedings of the International Conference on Artificial Intelligence, ICAI 2003, Los Angeles, USA, 2003.
[3] "The Fraunhoffer Resource Grid" Institut Arbeitschiwrtschaftund Otganization, Germany
[4] Sander, T. Peleschuk B., Grosz, A, "A Scalable Distributed Algorithm for Efficient Task Allocation", Proceedings of AAMAS' 02, Bologna, Italy, 2002.
[5] Dorigo, M., Middendorf, M., & Stutzle, T. (Eds.). (2000b), "Abstract Proceedings of ANTS 2000 – From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms", Brusseles, IRIDIA, Universite Libre de Bruxelles.