Resource Allocation and Management in Cloud Based Networks

T. Sathiyaseelan* & B. Anantharaj**

*Master of Engineering Student, Department of Computer Science and Engineering, Thiruvalluvar College of Engineering and Technology, Tamil Nadu, INDIA. E-Mail: seelan.sathiya@gmail.com

**Professor, Department of Computer Science and Engineering, Thiruvalluvar College of Engineering and Technology, Tamil Nadu, INDIA.

***Abstract*—**Resource allocation and management puts up special challenges in large scale cloud management systems like server clusters, grid computing of all types as per GRMS, which work on at the same time requests of a large number of customers. We concentrate primarily upon improvising the dynamic resource management in the large scale-cloud surroundings. Aggregating the request and ensuring the request are promptly collected from peers the request so collected from the clients and vice versa is important to estimate the size of the resource request to plan the allocation of resources, in a reverse scenario how to schedule the available resources against the job queue. Our core contribution - centre around outlining distributed middleware architecture and the presenting key elements, a gossip protocol P* which meets our 3 main design intentions: The resource allocation with respect to client hosted sites, efficient adaptation to load changes and, scalability in relation on the number of machines and sites. We present first a protocol which maximizes the cloud utility program under the central processing unit and memory restrictions and also minimizes the costs to adapt an allocation. Then we extend this protocol to have an administrative controlling parameter which can be done with the help of the profiling technique. The protocol continuously executes on dynamic, local input acquired through synchronization, as other proposed gossip protocols do. To develop a gossip protocol this is robust against node failures. In this paper we present P*, a gossip protocol for the durable supervision of accumulations which is robust against discontinuous failures.

***Keywords*—**Cloud Computing, Dynamic Resource Allocation, Gossip Protocol, Requirement Aggregation

***Abbreviations*—**Grid Resource Management Systems (GRMS), Simple Network Management Protocol (SNMP)

I. INTRODUCTION

IaaS solutions, such as OpenDRAC [www.opendrac.org], OpenNebula [http://www.opennebula.org], OpenStack [http://www.openstack.org] and Eucalyptus [http://www.eucalyptus.com], or PaaS solutions, such as AppScale [http://appscale.cs.ucsb.edu], WebSphere XD [http://www.ibm.com/software/webservers/appserv/extend/virtualenterprise/] and Cloud Foundry [http://www.cloudfoundry.com/]. These solutions include functions that compute placements of applications or virtual machines onto specific physical machines. However, they do not, in a combined and integrated form, (a) dynamically adapt existing placements in response to a change (in demand, capacity, etc.), (b) dynamically scale resources for an application beyond a single physical machine, (c) scale beyond some thousand physical machines (due to their centralized underlying architecture). These three features in integrated form characterize our contribution. The concepts in this paper thus outline a way to improve placement functions in these solutions. Regarding public clouds, little information is available with respect to the underlying capabilities of services provided by companies like Amazon [http://aws.amazon.com/ec2/], Google [http://code.google.com/appengine/] and Microsoft [http://www.microsoft.com/windowsazure/]. We expect that the road maps for their platforms include design goals for application placement which are similar to ours.

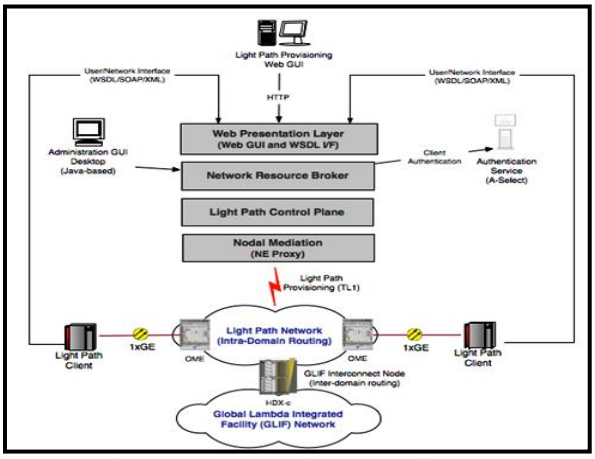

Figure 1 – DRAC Reference Architecture

The resource supervision is the process to acquire state information of a controlled system. In the traditional network and system management the supervision on a peer -device base is carried out by which an administrative station questions regularly devices in domain for the values of local variables, like device counter, achievement frame. Then these variables are worked on the administrative station to estimate an estimate of a net width of state which is analyzed and is traded by administrative programs. SNMP is probably the best of all known protocol which supports this type of the supervision which follows the pull approach. An alternative to the pull approach which is based on the polling of votes is the push approach by which net elements send values of local variables to the administrative station, whenever changes seem to those values. With the appearance of large and dynamic linked up systems the push approach gaining importance because this approach enables to the administrative system to follow lastingly the evolution of the net state. In this paper present up the monitoring problem, while we model the controlled system as a dynamic set of node $V(t)$ which represent the devices [Wuhib et al., 2007]. In the course of the time any number of nodes can join or leave the system, or they can fail [Jelasity et al., 2005; Pacifici et al., 2006].

Using an aggregation function, aggregation functions close SUM, extreme functions, histogram, as well as statistical means and moments. More in general, consider in our working aggregation functions which are replaceable as well as associatively in this setting; three classes of protocols are studied and contrived. The polling of votes of accumulations, the durable supervision of accumulations: the supervision of threshold crossings of accumulations [Dam & Stadler, 2006].

II. Related Works

The user group size is elastic in nature at any given time demanding different types of services and application according to their specific needs, a client application will run in each machine as a dynamic set of node $V(t)$ which represent the devices and all such requests are aggregated and updated into the global variable, which also helps us is assessing the total demand in any given point of time [Kempe et al., 2003; Tang & Ward, 2005].

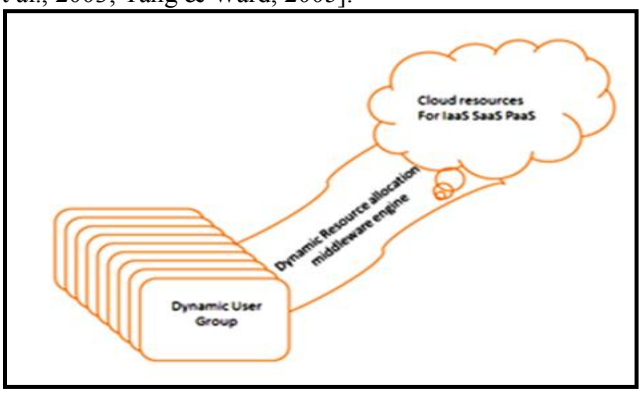

Figure 2 – Design Methodology

The middleware which will have the global variable O Centrally, our work is the supervision of a global variable (t) which is estimated by local variable $X_i(t)$ using an aggregation function, to update the global variable. Every machine will have a copy of machine manager component to estimates the resource allocation policy which encloses the decisions of the module instances to run. The resource allocation policy is estimated by a protocol which loads in resource allocation manager component. This component, takes like give, the respected inquiry for every module which leads the machine. The respected allocation policy becomes the module Scheduler for the fulfilment execution, as well as the site managers sent to meet decisions on the request mailing. The covering manager carries out a distributed algorithm which maintains a covering graph of the machines in the cloud and supplies every resource manager with a list of machines to work on each other. Our architecture combines a location or site manager with every location. It has two components: an inquiry profiler and a request forwarding agent. The inquiry profiler estimates the location or site enquiry of every module of the location being based on the request statistics. This resource demand estimate is sent on to all machine managers who load of modules which belong to this side. Similarly the request forwarding agent sends user requests to go to set of modules in a process which belong to this site. Module manager which sends on decisions, the resource allocation policy and restrictions considers like meeting affinity [Wuhib et al., 2010].

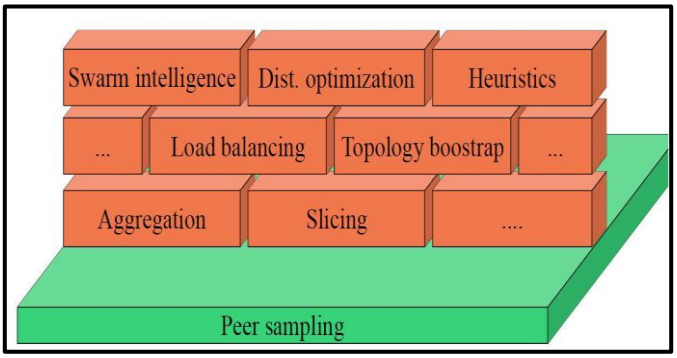

Figure 3 – Gossip Building Blocks

III. Proposed Model for Gossip Protocols

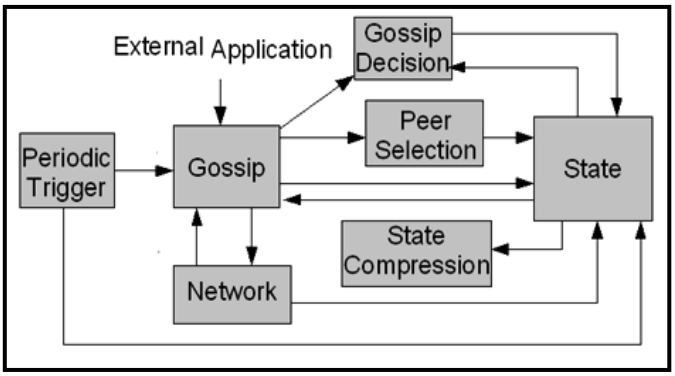

Figure 4 – Model for Gossip protocols

Figure 4 shows model for Gossip protocol, Under a Gossip algorithm, the operation at any node $i \in V$, must satisfy the following properties:

(1) The algorithm should only utilize information obtained from its neighbors $N(i) \triangleq \{j \in V : (i,j) \in E\}$.
(2) The algorithm performs at most $O(d_i\ logn)$ amount of computation per unit time.
(3) Let $|F_i|$ be the amount of storage required at node i to generate its output. Then the algorithm maintains $O(poly(logn) + |F_i|)$ amount of storage at node i during its running.
(4) The algorithm does not require synchronization between node i and its neighbors, $N(i)$.
(5) The eventual outcome of the algorithm is not affected by „reasonable" changes in $N(i)$ during the course of running of the algorithm.

3.1. Gossip Pseudo Code for Dynamic Resource Allocation

Algorithm 1 Protocol computes a configuration matrix A. Code for machine n

Initialization
read $\omega, \gamma, \Omega, \Gamma, row_n(A)$;
init Instance ();
start passive and active threads;
Active thread
for $r = 1\ to\ r_{max}$ do
n' = choosePeer();
send(n'; $row_n(A)$;); $row_{n'}(A)$; (A) = receive(n');
updatePlacement(n'; $row_{n'}(A)$);
sleep until end of round;
write $row_n(A)$;
Passive thread
while true do
$row_{n'}(A)$ = receive(n'); send(n'; $row_n(A)$);
updatePlacement(n'; $row_{n'}(A)$);

IV. SOURCE CODE

```
Public Resource get Endpoint Resource (User Details user Details, String resource ID) throws Exception
    {
    return dbResourceGroupProfile.getEndpointResource (resourceID);
    }
    Public GlobalPolicy getGlobalPolicy(UserDetails userDetails) throws Exception
    {
    return dbGlobalPolicy.getGlobalPolicy();
    }
    // See comments on getUserGroupNameLineage
    public List <String> getResourceGroupNameLineage(UserDetails userDetails,
    ResourceGroupProfile rgProfile, List<String> list) throws Exception
    { list.add(rgProfile.getName().toString());
    String createdByGroup = rgProfile.getMembership().getCreatedByGroupName();
    if (createdByGroup != null && createdByGroup.length() > 0)
    {
      try {
      ResourceGroupProfile parentProfile = dbResourceGroupProfile.getResourceGroupProfile(createdByGroup);
          if (!ALLOWlDISPLAY)
    {
          isAllowed (userDetails, newPolicyRequest (parentProfile, PolicyRequest.CommandType.READ));
    }
    getResourceGroupNameLineage(userDetails, parentProfile, list);
      } catch (RemotePolicyException e)
    {
        // Given the retrieval implementation, don't log here.
      }
    }
    else {
      Collections.reverse(list);
    }
    return list;
```

V. METHODS - GOSSIP ALGORITHMS

5.1. Separable Function Computation

Performance of the gossip algorithm for summation or equivalently separable function computation on various graph models [Mosk-Aoyama & Shah, 2006; Shah, 2008].

A. Complete Graph

For a complete graph of n nodes with natural symmetric probability matrix $P = [1/n]$, as explained in Preliminaries, $\Phi(P) = O(1)$. Therefore, $T_{sum}(\varepsilon, \delta) = O(\delta^{-2}\ logn)$ for $\varepsilon = \Omega(1/poly(n))$. That is, the algorithm performs computation essentially as fast as possible for any fixed δ.

B. Ring Graph

Consider the ring graph of n nodes with a natural symmetric probability matrix P, obtained by the Metropolis-Hasting method, i.e., for each i, $P_{ii} = 1/2, P_{ii^+} = P_{ii^-} = 1/4$ where i^+ and i^- represent neighbors of i on either side. As established in Preliminaries, $\Phi(P) = O(1/n)$ for such P. Therefore, $T_{sum}(\varepsilon, \delta) = O(\delta^{-2} n log n)$ for $\varepsilon = \Omega(1/poly(n))$. Since the diameter of a ring scales as n, this is again as fast as possible for any fixed δ.

C. Expander Graph

For a d-regular expander with all nodes having degree d, the natural P has $\Phi(P) = O(1)$. Therefore, like the complete graph $T_{sum}(\varepsilon, \delta) = O(\delta^{-2} log\ n)$ for $\varepsilon = \Omega(1/poly(n))$. That is, the algorithm performs computation essentially as fast as possible for any fixed δ.

D. Geometric Random Graph

For the geometric random graph over n nodes and connectivity radius $r = r(n)$ beyond the connectivity threshold, as established in Preliminaries, the natural probability matrix P on it has $\Phi(P)$ essentially scaling as r. Therefore, we will have $T_{sum}(\varepsilon, \delta) = O^8(\delta^{-2}r^{-1})$ for $\varepsilon = \Omega(1/poly(n))$. Again, since the diameter of $G(n,r)$ scales like $1/r$, the algorithm performs computation essentially as fast as possible for any fixed δ [Boyd et al., 2006].

5.2. Network Scheduling by using Gossip Protocol

Gossip based algorithm for scheduling nodes is compared with two well known network scenarios [Kempe & Kleinberg, 2002; Shah, 2008].

A. Multi-hop Queuing Network

For a given network G, let S be the set of data-flows with arrival rate λ_s for flow $s \in S$. Let f_s and d_s denote source and destination node respectively for flow $s \in S$. The routing is assumed to be pre-determined in the network. If s passes through $v \in V$ then let $h(v,s) \in V$ denote its next hop unless $v = d_s$ in which case its data departs from G. Let $Q_{vs}(t)$ denote the queue-size of flow s at node v at time t. Define,

$$W_{vs}(t) = \begin{cases} Q_{vs}(t) - Q_{h(v,s)s}(t), if\ v \neq d_s, \\ 0 \qquad\qquad\qquad if\ v = d_s. \end{cases}$$

Define $W_v(t) = max_{s \in S} W_{vs}(t)$ and $W(t) = [W_v(t)]$. Then the throughput optimal (stable) algorithm chooses $I^*(t)$ as the schedule which is a maximum weight independent set with respect to $W(t-1)$,

$$I^*(t) = \arg\max_{I \in t} \langle I, W(t-1) \rangle.$$

B. Joint Resource Allocation & Scheduling

It is very well explained that the problem of congestion control and scheduling decomposes into two weakly coupled sub-problems: (i) congestion control, and (ii) scheduling. We describe the link-level scheduling problem. We urge an interested reader to go through [Lin et al., 2006] for details. The setup of the problem is the same as in the previous example with difference that routing is not pre-determined. The coupling of congestion control and scheduling happens via Lagrange multipliers $q(t) = [q_e(t)]_{e \in E}$. With the interference model of this paper, the scheduling problem boils down to the selection of maximum weight independent set $I^*(t)$ with respect to weight $W(t-1) = [W_v(t-1)]$, where $W_v(t-1) = max_{e:e=(u,v) \in E} q_e(t-1)$.

VI. RESULTS

Performance of Gossip algorithms is strongly dependent on the spectral properties of underlying network graph. Accumulations and aggregations to eliminate the node failures and discontinuous node failures, requires an application highly robust in nature which can analyse and work in combination to maximise the performance thus reducing the failures [Birman, 2007].

VII. CONCLUSION AND FUTURE WORK

This paper, we perform a significant contribution to the technology of a resource management middleware for cloud surroundings. We identify a key component of such middleware and Gossip protocol which can be used to meet our design intentions to the resource management: Beauty of the resource allocation in relation on the part of, efficient adaptation to load changes and scalability of the middleware layer in relation on both the number of dynamic user request in the cloud as well as the number of sites had to hosted [Prieto & Stadler, 2007]. We presented a gossip protocol P* which, in a distributed and durable fashion, a heuristic solution to the resource allocation problem for a dynamically changing resource inquiry counts.

Future work will be of having an application with TCP stack at certain parts and using Gossip in rest of the area achieving a great balance and speed to meet the growing needs in resource allocation in cloud computing.

REFERENCES

[1] D. Kempe & J. Kleinberg (2002), "Protocols and Impossibility Results for Gossip-based Communication Mechanisms", *Proceedings of 43rd Annual IEEE Symposium Foundations Computer Science*, Pp. 471.

[2] D. Kempe, A. Dobra & J. Gehrke (2003), "Gossip-based Computation of Aggregate Information", *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, *IEEE Computer Society*, Pp. 482.

[3] M. Jelasity, A. Montresor & O. Babaoglu (2005), "Gossip-based Aggregation in Large Dynamic Networks", *ACM Transactions on Computer Systems*, Vol. 23, No. 3, Pp. 219–252.

[4] M. Dam & R. Stadler (2005), "A Generic Protocol for Network State Aggregation", *Proceedings of Radio Vetenskapoch Kommunikation (RVK)*.

[5] C. Tang & C. Ward (2005), "GoCast: Gossip-enhanced Overlay Multicast for Fast and Dependable Group Communication", *Proceedings of International Conference on Dependable Systems Networks (DSN*.

[6] S. Boyd, A. Ghosh, B. Prabhakar & D. Shah (2006), "Randomized Gossip Algorithms", *IEEE/ACM Transaction on Networking*, Vol. 14, No. SI, Pp. 2508–2530.

[7] G. Pacifici, W. Segmuller, M. Spreitzer & A. Tantawi (2006), "Dynamic Estimation of CPU Demand of Web Traffic", *Proceedings of the 1st International Conference on Performance Evaluation Methodologies and Tools*, New York, NY, USA: ACM, Pp. 26.

[8] D. Mosk-Aoyama & D. Shah (2006), "Computing Separable Functions via Gossip", *Proceedings of 25th Annual ACM Symposium Principles Distributed Computing (PODC)*.

[9] X. Lin, N. Shroff & R. Srikant (2006), "A Tutorial on Cross-layer Optimization in Wireless Networks", *Available through csl.uiuc.edu/rsrikant*.

[10] F. Wuhib, M. Dam, R. Stadler & A. Clemm (2007), "Robust Monitoring of Network-Wide Aggregates through Gossiping", *Proceedings of Integrated Manage (IM)*, Pp. 226–235.

[11] A.G. Prieto & R. Stadler (2007), "A-GAP: An Adaptive Protocol for Continuous Network Monitoring with Accuracy Objectives", *IEEE Transactions on Network and Service Management*, Vol. 4, Pp. 2–12.
[12] K. Birman (2007), "The Promise, and Limitations of Gossip Protocols", *ACM SIGOPS Operating Syst. Review Archive*, Vol. 41, No. 5.
[13] D. Shah (2008), "Foundations and Trends in Networking", Vol. 3, No. 1 Pp. 1–125.
[14] F. Wuhib, R. Stadler & M. Spreitzer (2010), "Gossip-based Resource Management for Cloud Environments (Long Version)", *KTH Royal Institute of Technology, Tech. Report*.
[15] OpenNebula Project Leads, http://www.opennebula.org/, Feb. 2012.
[16] OpenStack LLC, http://www.openstack.org, Feb. 2012.
[17] Eucalyptus Systems, Inc., http://www.eucalyptus.com/, Feb. 2012.
[18] UC Santa Barbara, http://appscale.cs.ucsb.edu/, Feb. 2012.
[19] "IBM WebSphere Application Server," http://www.ibm.com/software/webservers/appserv/extend/virtualenterprise/, Feb. 2012.
[20] VMWare, http://www.cloudfoundry.com/, Feb. 2012.
[21] Amazon Web Services LLC, http://aws.amazon.com/ec2/, Feb. 2012.
[22] Google Inc., http://code.google.com/appengine/, Feb. 2012.
[23] Microsoft Inc., http://www.microsoft.com/windowsazure/, Feb. 2012.
[24] OpenDRAC., https://www.opendrac.org/

T. Sathiyaseelan, First Author (Correspondence Author) – MCA, ME (Pursing) Thiruvalluvar College of Engineering and Technology.
B. Anantharaj, Second Author – ME, Professor, Thiruvalluvar College of Engineering and Technology.