International Journal of Computer and Communication Technology

Volume 7 | Issue 4 Article 15

October 2016

DYNAMIC RESOURCE MANAGEMENT IN LARGE CLOUD ENVIRONMENTS USING DISTRIBUTED GOSSIP PROTOCOL

M. SASITHARAGAI
Department of Computer Science & Engineering , Angel College of Engineering and Technology, Tirupur-India., sasitharagai@gmail.com

A. PADMASHREE
Department of Computer Science and Engineering, Angel College of Engineering and Technology, Tirupur, apadmashree.me@gmail.com

T. DHANALAKSHMI
Department of Computer Science and Engineering, Angel College of Engineering and Technology, Tirupur, dhanaesec@gmail.com

S. GOWRI
Department of Computer Science & Engineering , Angel College of Engineering and Technology, Tirupur-India, gowri1666@gmail.com

DYNAMIC RESOURCE MANAGEMENT IN LARGE CLOUD ENVIRONMENTS USING DISTRIBUTED GOSSIP PROTOCOL

M.SASITHARAGAI[1], A.PADMASHREE[2], T.DHANALAKSHMI[3] & S.GOWRI[4]

[1,2,3&4]Department of Computer Science and Engineering, Angel College of Engineering and Technology, Tirupur
Email: sasitharagai, apadmashree.me, dhanaesec & gowri1666@ gmail.com

Abstract- Resource management poses particular challenges in large-scale systems, such as server clusters that simultaneously process requests from a large number of clients. We mainly focus on the dynamic resource management in large scale cloud environment. Our core contribution centers around outlining a distributed middleware architecture and presenting one of its key elements, a gossip protocol P* that meets our 3 main design goals: (1) fairness of resource allocation with respect to hosted sites (2) efficient adaptation to load changes and (3) scalability in terms of both the number of machines and sites. We first present a protocol that maximizes the cloud utility under CPU and memory constraints and also minimizes the cost for adapting an allocation. Then, we extend that protocol to have a management control parameter, which can be done with the help of profiling technique. A particular challenge is to develop a gossip protocol that is robust against node failures. In this paper, we present P*, a gossip protocol for continuous monitoring of aggregates, which is robust against discontiguous failures (i.e., under the constraint that neighboring nodes do not fail within a short period of each other)

Index Terms *- Cloud computing, distributed management, resource allocation, gossip protocol, profiling, robust aggregation*

I. INTRODUCTION

We focus the problem of resource management for a large-scale cloud environment. Such an environment includes the physical infrastructure and associated control functionality that enables the provisioning and management of cloud services.

Dynamic resource management can be explained with the help of IaaS and PaaS perspectives. The perspective we take is that of a cloud service provider [1], which hosts sites in a cloud environment and the stakeholders for this use case are depicted in Fig 1.

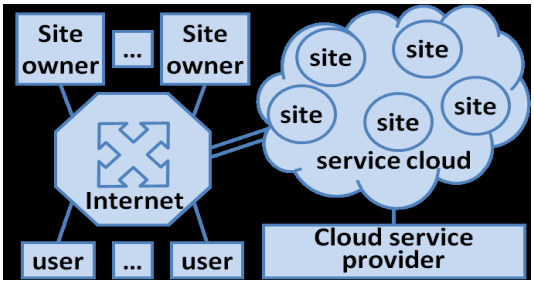

Fig 1. Deployment scenario with the stakeholders of the cloud environment considered in this work

This work [2] contributes towards engineering a middleware layer that performs resource allocation in a cloud environment, with the following design goals:
1) Performance objective: We consider computational and memory resources and the objective is to achieve reasonable fairness among sites for computational resources under memory constraints.
2) Adaptability: The resource allocation process must dynamically and efficiently adapt to changes in the demand from sites.
3) Scalability: The resource allocation process must be scalable both in the number of machines in the cloud and the number of sites that the cloud hosts. This means that the resources consumed per machine in order to achieve a given performance objective must increase sub linearly with both the number of machines and the number of sites.

This paper also addresses a fundamental problem in virtual machine (VM) resource management [3]: how to effectively profile physical resource utilization of individual VMs. Our focus is on extracting the utilization of physical resources by a VM across time, where the resources include CPU (utilization in CPU cycles), memory (utilization in memory size). Correct VM resource utilization information is tremendously important in any autonomic resource management that is model based. Hence, resource management is completely based on resource mapping across virtual machines.

Profiling is a hard problem because mapping virtual-to-physical (V2P) resource activity mapping is not always one to one and may depend on application workload characteristics. In this paper we extend the factor graph model [5] with directionality and factoring generalization, and design a directed factor graph (DFG) that models the multivariate dependence relationships among different resources and across virtual and physical layers.

Gossip protocols, also known as epidemic protocols [12], can be characterized by asynchronous and often randomized communication among nodes in a network [7]. Originally, they have been proposed for disseminating information in large dynamic

environments and more recently, they have been applied for various tasks, including constructing robust overlays, estimating the network size [8] [6], etc.

A gossip protocol [4] for monitoring network-wide aggregates executes in the context of decentralized management architecture. Fig 2. shows an example of such an architecture, which we propose using for this purpose. In this architecture, monitoring nodes with identical functionality organize themselves into a management overlay. The aggregation protocol (in this case, the gossip protocol) runs in the monitoring nodes, which communicate via the overlay. Each monitoring node collects data from one or more network devices. A management station can access the monitoring layer at any node. Node or link failures—on the physical network or the management overlay—trigger a re-organization of the management overlay, thereby enabling continuous operation.

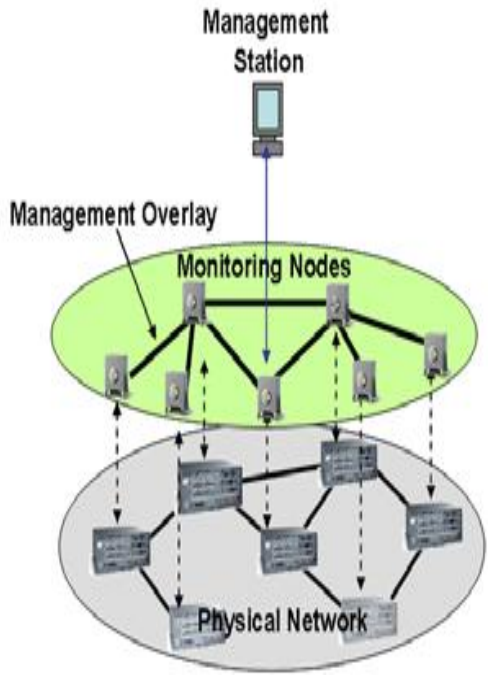

Fig 2. Architecture of the decentralized monitoring system. Gossip protocols run in the management overlay (middle layer)

Hence, gossip protocol can be formally applied to dynamic resource management in large cloud environments with a focus of overcoming discontiguous failures and also using a management control parameter for the protocol actions to take place securely.

II. SYSTEM ARCHITECTURE

A cloud environment spans several datacenters interconnected by an internet. Each of these datacenters contains a large number of machines that are connected by a high-speed network. Users access sites hosted by the cloud environment through the public Internet. A site is typically accessed through a URL that is translated to a network address through a global directory service, such as DNS. A request to a site is routed through the Internet to a machine inside a datacenter that either processes the request or forwards it.

Fig 3. (left) shows the architecture of the cloud middleware [1] [2]. The components of the middleware layer run on all machines. The resources of the cloud are primarily consumed by module instances whereby the functionality of a site is made up of one or more modules.

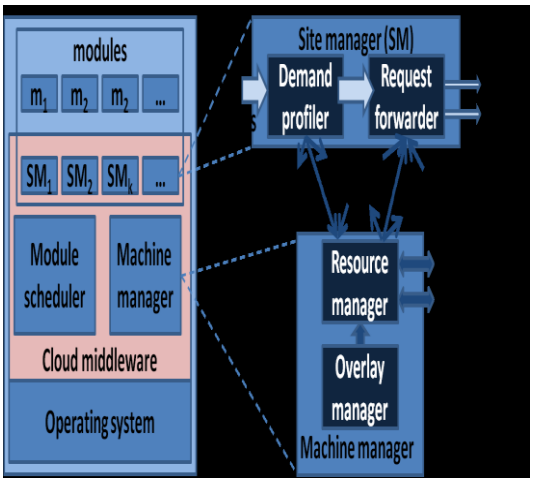

Fig 3. The architecture for the cloud middleware (left) and components for request handling and resource allocation (right).

Each machine runs a *machine manager* component that computes the resource allocation policy, which includes deciding the module instances to run. The resource allocation policy is computed by a protocol P* that runs in the *resource manager* component. This component takes as input the estimated demand for each module that the machine runs. The computed allocation policy is sent to the *module scheduler* for execution, as well as the *site managers* for making decisions on request forwarding. The *overlay manager* implements a distributed algorithm that maintains an overlay graph of the machines in the cloud and provides each resource manager with a list of machines to interact with.

Our architecture associates one site manager with each site. A site manager handles user requests to a particular site. It has two components: (1) The *demand profiler* estimates the resource demand of each module of the site based on request statistics. This demand estimate is forwarded to all machine managers that run instances of modules belonging to this site. (2) The *request forwarder* sends user requests for processing to instances of modules belonging to this site. Request forwarding decisions take into account the resource allocation policy and constraints such as session affinity. Fig 3. (right)

shows the components of a site manager and how they relate to machine managers.

Profiling [3] can be done with the help of Directed Factor Graph (DFG) which can be explained with the help of an example as shown below.

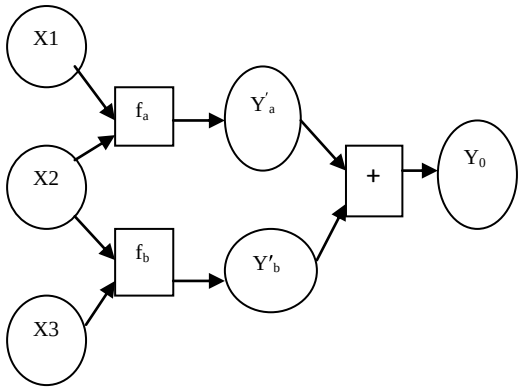

Fig 4. A directed factor graph example

Fig 4. shows the directed factor graph for a global function $Y_0 = g(x1, x2, x3)$ with decomposition given as,

$g(x1, x2, x3) = f_a(x1, x2) + f_b(x2, x3)$

The new variable nodes Y'_a, Y'_b are two temporary variables recording the output of the functions f_a and f_b.

DFG can be used for resource mapping across virtual machines, which serves as the management control parameter, so as to increase the efficiency of gossip protocol. Also, the choice of using gossiping is motivated by achieving robustness [4]. In case a parent node fails, an elected child node can replace the parent instantly. The goal for our protocol is to give a continuous estimate of a global aggregate in near real-time and with high accuracy and it address the problem of mass loss in gossip protocols.

III. FORMALIZING THE PROBLEM OF RESOURCE ALLOCATION AND PROFILING BY THE ROBUST GOSSIP PROTOCOL

Hence, formalizing the resource allocation problem in dynamic large cloud environment is a tedious process but making this process to be efficient, we need a robust gossip protocol to perform dynamic resource management by satisfying performance objectives.

For this work, we consider a cloud as having computational resources (i.e., CPU) and memory resources, which are available on the machines in the cloud infrastructure [1] [2]. We formulate the resource allocation problem as that of maximizing the cloud utility under CPU and memory constraints [14]. The solution to this problem is a configuration matrix that controls the module scheduler and request forwarder components. At discrete points in time, events occur, such as load changes, addition and removal of site or machines, etc. In response to such an event, the optimization problem is solved again, in order to keep the cloud utility maximized. We add a secondary objective to the optimization problem, which states that the cost of change from the current configuration to the new configuration must be minimized.

For the above model, we consider a cloud with CPU capacity $\mathbf{\Omega}$, memory capacity $\mathbf{\Gamma}$, and demand vectors $\boldsymbol{\omega}$, $\boldsymbol{\gamma}$. We first discuss a simplified version of the problem. It consists of finding a configuration A that maximizes the cloud utility U^c:

$$\begin{aligned} &\text{maximize} && U^c(A,\boldsymbol{\omega}) \\ &\text{subject to} && A \geq 0,\ \mathbf{1}^{\mathbf{T}}\mathbf{A} = \mathbf{1}^{T} \\ & && \Omega^{\wedge}(A,\boldsymbol{\omega})\mathbf{1} \leq \mathbf{\Omega} \end{aligned}$$

Our concept of utility is max-min fairness and our goal is to achieve fairness among sites. This means that we want to maximize the minimum utility of all sites, which we achieve by maximizing the minimum utility of all module instances.

Second, we consider the fact that the system must adapt to external events in order to keep the cloud utility maximized. Therefore, the problem becomes one of adapting the current configuration $A(t)$ at time t to a new configuration $A(t + 1)$ at time $t + 1$ which achieves maximum utility at minimum cost of adapting the configuration.

$$\begin{aligned} &\text{maximize} && U^C(A(t+1), \boldsymbol{\omega}(t+1)) \\ &\text{minimize} && c^*(A(t), A(t+1)) \\ &\text{subject to} && A(t+1) \geq 0,\ \mathbf{1}^{\mathbf{T}}\mathbf{A}(t+1) = \mathbf{1}^{\mathbf{T}} \\ & && \Omega^{\wedge}(A(t+1),\boldsymbol{\omega}(t+1))\mathbf{1} \leq \mathbf{\Omega} \\ & && \mathbf{sign}(A(t+1))\boldsymbol{\gamma} \leq \mathbf{\Gamma}. \end{aligned}$$

P* is an asynchronous protocol. This means that a machine does not synchronize the start time of a protocol round with any other machine of the cloud. At the beginning of a round (more precisely, at the start of the loop of the active or passive thread), a machine reads the current demands of the modules it runs. At the end of a round (more precisely, at the end of the loop of the active or passive thread) a machine updates its part of the configuration matrix A. The matrix A thus changes dynamically and asynchronously during the evolution of the system [13].

The work in [9], which has been extended by [10] presents a distributed middleware for application placement in datacenters. As in this paper, the goal of that work is to maximize a cluster utility under changing demand, although a different concept of utility is used. The choice of utility functions in that work is such that service differentiation works

very well under overload conditions, with the risk of starving unpopular applications. In contrast, our approach guarantees that every module receives its fair share of the CPU resources of the cloud, and that in underload conditions all modules are guaranteed to

have satisfied demands. The proposed design in [9], [10] scales with the number of machines, but it does not scale in the number of applications.

The modeling process of DFG [3] consists of the following steps:
1. Host a single VM in a server.
2. Run a benchmark for a specific virtual resource (e.g., a CPU-intensive benchmark).
3. Apply statistics analysis to find out the set of physical resources on which the benchmark incurs non-negligible utilization and learn the models for the function nodes.
The benchmark based modeling process [11] aims at capturing the stable causality relationships between virtual and physical resource demands. We carefully select a fixed set of benchmark applications to cover the two resources (CPU and memory)

Then, our main aim is to develop a distributed protocol for continuously computing aggregation functions in a scalable and robust manner [4]. G-GAP is based on "Push-Synopses", a gossip protocol for computing aggregates proposed by Kempe et al. [7]. Here we consider Push-Synopses applied only to the computation of averages, although we don't envisage any problems in adapting our results to more general synopses, such as those discussed in [7]. Our main contribution in this paper is to extend the Push-Synopses protocol with a scheme to provide accurate estimates in the event of node failures of different types. These extensions are introduced in two steps; first, for the case of fully synchronized rounds with guaranteed, timely message delivery; then, for the more general, asynchronous case.

In the Push-Synopses protocol, each node *i* maintains, in addition to the local management variable x_i , a *weight* w_i and a *sum* s_i . The local estimate of the aggregate is computed as $a_i = s_i / w_i$. The protocol is given for the case of a complete (i.e. fully connected) network graph of *n* nodes. However, the protocol is easily adapted to graphs where only adjacent nodes are allowed to communicate directly with each other. This is the relevant case in practice, for scalability reasons. The protocol executes in synchronized rounds, assuming reliable and timely communication, such that a message sent within a given round is guaranteed to be delivered within that round.
The protocol relies on five rather strong assumptions which makes it robust:

1. *Reliable and timely message delivery*: There is a maximum communication delay $t_d < t_r$ (the roundduration) such that a message sent from a node *i* to a node *j* at time *t* is delivered to *j* no later than $t + t_d$
2. *Synchronized rounds*: Rounds are globally synchronized to within some bound $t_{\Delta r}$. That is, all live nodes start a round within $t_{\Delta r}$ of each other.
3. *Round atomicity*: All protocol cycles are executed as atomic statements.
4. *Discontiguous crash failures*: No two nodes fail within two rounds of each other. When running this protocol on a network graph, this assumption translates to condition that adjacent nodes cannot fail within a period of two rounds.
5. *Connectedness*: No failure will cause a node to become disconnected.

IV. DISCUSSION AND CONCLUSION

With this paper, we make a significant contribution towards engineering a resource management middleware for a site-hosting cloud environment. We identify a key component of such a middleware and present a protocol that can be used to meet our design goals for resource management: fairness of resource allocation with respect to sites, efficient adaptation to load changes and scalability of the middleware layer in terms of both the number of machines in the cloud as well as the number of hosted sites.

Also, we present the design and evaluation of a VM monitoring information calibration mechanism. We formulate our problem as a source separation problem and base our solution on a directed factor graph. We show how to build a base DFG model through benchmarking and design a run-time remodeling solution which is adaptive and guided by the base DFG model. Our evaluation shows that the proposed methodology is robust as it successfully calibrates the VM monitoring information and compares well to baseline measures.

This paper also makes a major contribution by presenting a gossip protocol, P*, which enables continuous monitoring of network-wide aggregates. The hard part is making the protocol robust against node failures. Applying gossip protocols to continuous monitoring is not possible without solving the problem of mass loss due to node failures.

Pursuing this goal, we plan to address the following issues in future work: (1) Develop a distributed mechanism that efficiently places new sites. (2) Extend the middleware design to span several clusters and several datacenters, while keeping module instances of the same site "close to each other", in order to minimize response times and communication overhead. (3) Extend P* to allow the memory demand to change over time. (4) Extend P* to consider additional resource types, such as storage and network resources.

V. REFERENCES

[1] F. Wuhib, R. Stadler, and M. Spreitzer, "Gossip-based resource management for cloud environments," in *2010 International Conference on Network and Service Management*.

[2] Fetahi Wuhib, Rolf Stadler and Mike Spreitzer, " A Gossip Protocol for Dynamic Resource Management in Large Cloud Environments" IEEE Trans. Network and Service Management, vol. 9, no. 2, pp. 213-225, June 2012.

[3] Lei Lu, Hui Zhang, "Untangling Mixed Information to Calibrate Resource Utilization in Virtual Machines" in *ICAC'11,* June 14–18, 2011, Karlsruhe, Germany.

[4] F. Wuhib, M. Dam, R. Stadler, and A. Clem, "Robust monitoring of network-wide aggregates through gossiping," IEEE Trans. Network and Service Management, vol. 6, no. 2, pp. 95–109, June 2009.

[5] V. M. Koch. *A Factor Graph Approach to Model-Based Signal Separation*. Hartung-Gorre Verlag Konstanz, First edition. 328 pages. ISBN-10: 3-86628-140-4., 2007.

[6] A. Ghodsi, S. El-Ansary, S. Krishnamurthy, and S. Haridi, "A Selfstabilizing Network Size Estimation Gossip Algorithm for Peer-to-Peer

Systems," SICS Technical Report T2005:16, 2005.

[7] D. Kempe, A. Dobra and J. Gehrke, "Gossip-Based Computation of Aggregate Information," In Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03), Cambridge, MA, USA, October 11-14, 2003.

[8] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, A. Demers, "Decentralized Schemes for Size Estimation in Large and Dynamic Groups," In proc. of the 4th IEEE International Symposium on Network Computing and Applications (NCA'05), Cambridge, MA, USA, July 27-29, 2005.

[9] C. Adam and R. Stadler, "Service middleware for self-managing large scale systems," *IEEE Trans. Network and Service Management, vol. 4*, no. 3, pp. 50–64, Apr. 2008.

[10] J. Famaey, W. De Cock, T. Wauters, F. De Turck, B. Dhoedt, and P. Demeester, "A latency-aware algorithm for dynamic service placement in large-scale overlays," in *2009 International Conference on Integrated Network Management*.

[11] C. Isci, J. E. Hanson, I. Whalley, M. Steinder, and J. O. Kephart. Runtime Demand Estimation for Effective Dynamic Resource Management. In *NOMS'10*, pages 381–388, 2010.

[12] A. Demers, D. Green, C. Hauser, W. Irish, J. Larson, "Epidemic algorithms for replicated database maintenance," In proc. the 6th Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10 - 12, 1987.

[13] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi, "Dynamic estimation of CPU demand of web traffic," in valuetools '06: Proceedings of the 1st international conference on Performance evaluation methodolgies and tools. New York, NY, USA: ACM, 2006, p. 26.

[14] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade, "Utility-based placement of dynamic web applications with fairness goals," in Network Operations and Management Symposium, 2008. NOMS 2008. IEEE, april 2008, pp. 9 –16.