ISSN: 2321-7782 (Online)
Volume 2, Issue 2, February 2014
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Paper / Case Study
Available online at: www.ijarcsms.com

Dynamic Resource Allocation in Large Cloud Environments using Gossip Protocol

J. Jayasakthi[1]
Dept of CSE
PRIST University
Tamilnadu – India

Dr. S. Thirunirai Senthil[2]
Professor
PRIST University
Tamilnadu – India

Abstract: ***Cloud computing is a term, which involves virtualization, distributed computing, networking, software and web services. A cloud consists of several elements such as clients, datacenter and distributed servers. It includes fault tolerance, high availability, scalability, flexibility, reduced overhead for users, reduced cost of ownership, on demand services etc. Gossip Protocol is effective protocol for the dynamic load balance in the distributed system and continuously executes process input & output process. Gossip protocol ensures fair resource allocation among sites/applications; it dynamically adapts the allocation to load changes and scales both in the number of physical machines and sites/applications. Gossip protocol avoids the resource allocation problem as that of dynamically maximizing the cloud utility under CPU and memory constraints. we extend that protocol to provide an efficient heuristic solution for the complete problem, which includes minimizing the cost for adapting an allocation. The protocol continuously executes on dynamic, local input and does not require global synchronization, as other proposed gossip protocols do.***

Keywords: ***Cloud computing, resource allocation, gossip protocols, distributed management.***

I. INTRODUCTION

Cloud computing is a model that treats the resources on the Internet as a unified entity, a cloud. Consider 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. From the perspective of the Platform-as-a-Service (PaaS) concept, with the specific use case of a cloud service provider which hosts sites in a cloud environment. This paper introduces resource allocation protocol that dynamically places site modules (or virtual machines, respectively) on servers within the cloud, following global management objectives.

Technically speaking, this work 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 maxmin fairness among sites for computational resources under memory constraints. Under this objective, each site receives CPU resources proportional to its CPU demand.

(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 (by the process) per machine in order to achieve a given performance objective must increase sublinearly with both the number of machines and the number of sites.

The core contribution of the paper is a gossip protocol P*, which executes in a middleware platform and meets the design goals outlined above. The protocol has two innovative characteristics. First, while gossip protocols for load balancing in distributed systems have been studied before, no results are available for cases that consider memory constraints and the cost of reconfiguration, which makes the resource allocation problem hard to solve. In this paper, we give an optimal solution for a simplified version of the resource allocation problem and an efficient heuristic for the hard problem.Second, the protocol we propose continuously executes, while its input and consequently its output dynamically changes.Most gossip protocols that have been proposed to date are used in a different way. They assume static input and produce a single output value.Whenever the input changes, they are restarted and produce a new output value,which requires global synchronization. protocol P*, which continuously executes and dynamically solves the problem of optimally placing applications in a cloud, achieving fair resource allocation.

II. System Architecture

Datacenters running a cloud environment often contain 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 the datacenter that either processes the request or forwards it.

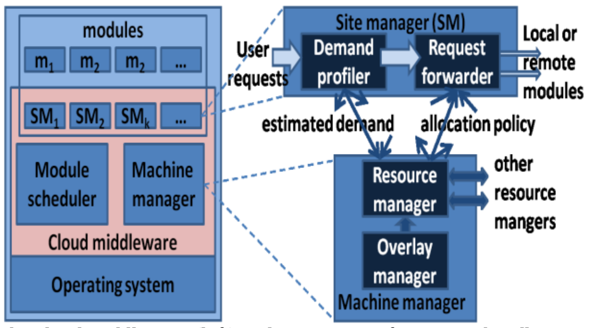

Fig 1. The architecture for the cloud middleware (left) and components for request handling and resource allocation (right)

Figure 1 (left) shows the architecture of the cloud middleware. 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. In the middleware, a module either contains part of the service logic of a site (denoted by mi in Figure 1) or a site manager (denoted by SMi).

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 (later in the paper called 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 implementation/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: a demand profiler and a request forwarder.

The demand profiler estimates the resource demand of each module of the site based on request statistics, QoS targets, etc. This demand estimate is forwarded to all machine managers that run instances of modules belonging to this site.

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. Figure 1 (right) show the components of a site manager and how they relate to machine managers.

The above architecture is not appropriate for the case where a single site manager can not handle the incoming request stream for a site. However, a scheme for a site manager to scale can be envisioned. For instance, a layer 4/7 switch could be introduced that splits the load among several instances of site managers, whereby each such instance would function like a site manager associated with a single site.

III. A Protocol for Distributed Resource Allocation

In this section, we present our protocol for resource allocation in a cloud environment, which we call P*. It is based on a heuristic algorithm for solving optimization problem and is implemented in form of a gossip protocol.

As a gossip protocol, P* has the structure of a round based distributed algorithm (whereby round-based does not imply that the protocol is synchronous). When executing a round-based gossip protocol, each node selects a subset of other nodes to interact with, whereby the selection functions is often probabilistic. Nodes interact via 'small' messages, which are processed and trigger local state changes. Node interaction with P* follows the so-called push-pull paradigm, whereby two nodes exchange state information, process this information and update their local states during a round. Compared to alternative distributed solutions, gossip-based protocols tend to be simpler, more scalable and more robust.

P* runs on all machines of the cloud. More precisely, it executes in the resource manager components of the middleware architecture. At the time of initialization, the resource manager implements a feasible cloud configuration A. After that, it invokes P* to compute and dynamically adapt the configuration with the goal to optimize the cloud utility (which means achieving max-min fairness among sites). Whenever the protocol has computed a new configuration, encoded in the matrix A, the resource manager checks whether the gain in utility of the newly computed configuration over the currently implemented configuration outweighs the cost of realizing the change. If this is the case, then the resource manager implements the new configuration encoded in A. The protocol P* takes as input the available cloud resources, the current configuration A and the current resource demand. It further relies on a set of candidate machines to interact with a given machine. This set is produced and maintained by the overlay manager component of the machine manager.

P* is designed to run continuously in an asynchronous environment where a machine does not synchronize the start time of a protocol round with any other machine. Further, a machine coordinates an update of the configuration A only with one additional machine at a time, namely its current interaction partner in the gossip protocol. Therefore, during the evolution of the system, the implemented cloud configuration A changes dynamically and asynchronously, as each machine maintains its part of the configuration. (Note that, to be precise, we use A in this section in two ways: (1) as the output of the protocol P* and (2) as the implemented cloud configuration.)

IV. Conclusion and Future Work

In this paper, we presented a gossip protocol P* that computes, in a distributed and continuous fashion, a heuristic solution to the resource allocation problem for a dynamically changing resource demand. Pursuing this goal, we plan to address the following issues in future work: (1) Develop a distributed mechanism that efficiently places new sites. (A mechanism for removing sites is straightforward, since P* will reallocate the freed-up resource.) (2) Extend the middleware design to become robust to various types of failures. (3) Extend the middleware design to span several clusters and several datacenters.

With respect to the protocol P*, we plan to investigate the following issues, in addition to the ones mentioned above. (1) Extend P* with a management control parameter that allows a management system to dynamically tune the CPU allocation to sites/applications. (2) Identify suitable decision functions that control the tradeoff between maximizing utility and minimizing cost of reconfiguration. (3) Investigate how close a configuration computed by P* is to the optimal solution to optimization problem. (4) Extend P* to allow the memory demand to change over time. (5) Extend P* to consider additional resource types, such as storage and network resources.

References

1. M. Jelasity, A. Montresor, and O. Babaoglu, “Gossip-based aggregationin large dynamic networks,” ACM Trans. Computer Syst., vol. 23, no. 3, pp. 219–252, 2005.
2. M. Jelasity, A. Montresor, and O. Babaoglu, “T-Man: gossip-based fast overlay topology construction,” Computer Networks, vol. 53, no. 13, pp. 2321–2339, 2009.
3. F. Wuhib, R. Stadler, and M. Spreitzer, “Gossip-based resource management for cloud environments,” in 2010 International Conference on Network and Service Management.
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. F. Wuhib, M. Dam, and R. Stadler, “A gossiping protocol for detecting global threshold crossings,” IEEE Trans. Network and Service Management, vol. 7, no. 1, pp. 42–57, Mar. 2010.
6. R. Yanggratoke, F. Wuhib, and R. Stadler, “Gossip-based resource allocation for green computing in large clouds,” in 2011 International Conference on Network and Service Management.