IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

A Dynamic and Efficient Resource Management by using Heuristics Algorithm

Anandharaj.V*1, Ms.A.Geetha2
*1,2Bharath University, Chennai, India
anandharajv.ece@gmail.com

Abstract

Dynamic resource management for a large-scale cloud environment is problematic one. We propose a gossip protocol that ensures fair resource allocation among sites/applications, dynamically adapts the allocation to load changes and scales both in the number of physical machines and sites/applications. We present a protocol that computes an optimal solution without considering memory constraints and prove correctness and convergence properties. Next, 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: In this paper, a mechanism is based on gossip protocol, heuristic algorithm.

Introduction

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. Cloud computing deliver the computing as a services whereby share resources, software, information via Internet which are accessed by the browser. The business software and data are stored in server at Remote Location (CLOUD), Cloud computing provides the kinds of services that are Infrastructure, Software, platform as a services. Gossip Protocol is effective protocol for the dynamic load balance in the distributed system and continuously execute process input & output process. Resources allocation policies are computed by protocols. Our contribution includes outlining distributed middleware architecture and presenting one of its key elements: a gossip protocol that (1) ensures fair resource allocation among sites/applications, (2) dynamically adapts the allocation to load changes and (3) scales both in the number of physical machines and sites/applications. The protocol continuously executes on dynamic, local input and does not require global synchronization, as other proposed gossip protocols

Existing System

Existing System:

Application placement in datacenters is often modeled through mapping a set of applications onto a set of machines such that some utility function is maximized under resource constraints. This approach has been taken, and solutions from these works have been incorporated in middleware products. The problem of resource management is application placement and load balancing in processor networks.

Example:

The Service provides some resources to access for the customer, that the resources are access often by the client. The concurrent accessing the capability of bandwidth is to sequencly reduces. Often accesses the resources mean the client need to deal with the Service provider for meet the Service level Objective.

Existing System Algorithm

Distributed Load Balancing.

Definition:

Distributed dynamic load balancing can introduce immense stress on a system in which each node needs to interchange status information with every other node in the system. It is more advantageous when most of the nodes act individually with very few interactions with others. Load balancing in cloud computing systems is really a challenge now. Always a distributed solution is required. Because it is not always practically feasible or cost efficient to maintain one or more idle services just as to fulfill the required demands. Jobs can't be assigned to appropriate servers and clients individually for efficient load balancing as cloud is a very complex structure and components are present throughout a wide spread area. Here some uncertainty is attached while jobs are assigned

Drawbacks

- Application placement is difficult one.
- Cannot handle the heavy loads.
- Resource allocation not properly maintained.

Conclusion

Using this Distributed Load Balancing Algorithm is successful one but the hosting the client application and the User site are slightly unfamiliar to the user. Distributed system allocation not properly maintained , in case of heavy not able to balance but load balancing of cloud environment is really challenge one but that the algorithm is infeasible.

Future Enhancement

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.

Comparison of Proposed And Existing System

Existing System	Proposed System
Resource allocation handling is infeasible, the Existing system application placement is have less difficulties , the need to invoke the difficulties	Resource allocation handling is possible. Using protocols to maintain the scalablity of the hosted applications of the authority.
Distributed load balancing was used load. Unsuitable for Heavy Process .The benefit of a single, continuous execution with restarts is that global synchronization	Heuristic algorithm. Suitable for heavy processes. The benefit of a sequence continuous execution no need restarts is that global synchronization
Protocols are not used, the algorithm used in existing is not have the performance metric of protocols.	Gossip protocols are used. while gossip protocols for load balancing in distributed systems
Less contribution towards engineering a resource management middleware for cloud environments.	Significant contribution towards engineering a resource management middleware for cloud environments.
Infeasible to scale the machine that is connected in application current time	Feasible to scale the machine that is connected in the application at current time.
We focus on application placement: on which machine to place a specific module, and how to adapt placement decisions to change in demand	The cloud service provider operates the physical infrastructure. The cloud hosts sites belonging to its clients. Users access sites through the Internet. A site is composed of modules.
Infeasible to adapt change and load balancing at runtime. There problem of resource management application placement and load balancing in processor networks	Feasible to adapt changes and load balancing at runtime. Feasible of resource management is application placement and load balancing in processor networks

Proposed System

Abstract

Dynamic resource management for a large-scale cloud environment is problematic one. We propose a gossip protocol that ensures fair resource allocation among sites/applications, dynamically adapts the allocation to load changes and scales both in the number of physical machines and sites/applications. We present a protocol that computes an optimal solution without considering memory constraints and prove correctness and convergence properties. Next, 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.

Proposed System

In proposed system, we make a significant contribution towards engineering a resource management middleware for cloud environments. 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. We present 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. For instance, regarding fairness, the protocol performs close to an ideal system for scenarios where the ratio of the total memory capacity to the total memory demand is large.

Over All Diagrams

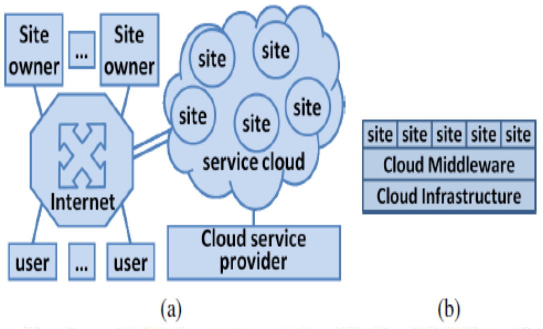

Fig. 1. (a) Deployment scenario with the stakeholders of the cloud environment considered in this work. (b) Overall architecture of the cloud environment; this work focuses on resource management performed by the middleware layer.

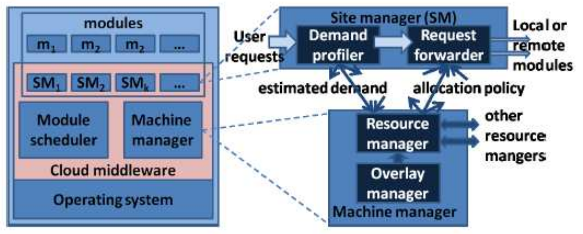

Fig. 2. The architecture for the cloud middleware (left) and components for request handling and resource allocation (right).

Fig 1.2 Proposed System Diagram

Scope of the Project

Our contribution includes outlining a distributed middleware architecture and presenting one of its key elements: a gossip protocol that ensures fair resource allocation among sites/applications, dynamically adapts the allocation to load changes and scales both in the number of physical machines and sites/applications. 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 core contribution of the paper is a gossip protocol, which executes in a middleware platform and meets the design goals. The protocol has two innovative characteristics. First, while gossip protocols for load balancing in distributed systems have been studied before, (to our knowledge) no results are available for cases that consider memory constraints and the cost of reconfiguration, which makes the resource allocation problem hard to solve.
This paper introduces a resource allocation protocol that dynamically places site modules (or virtual machines, respectively) on servers within the cloud, 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 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.

Example:

Step 1: Gossip protocol has the functionality to scale the all machine and fair ` resource allocation and dynamically adapt the load changes.

Step 2: Gossip protocol which executes in a middleware platform

Step 3: Each machine runs a machine manager component that computes the resource allocation policy, which includes deciding the module instances to run

Step 4: Each site has one site manager. A site manager handles user requests to a particular site.

Step 5: The demand profiler estimates the resource demand of each module of the site based on request statistics, QoS targets. This demand estimate is forwarded to all machine managers that run instances of modules belonging to this site

Step 6: 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.

Step 7: single site manager cannot 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.

Proposed System Algorithm Explanation

Heuristic algorithm.

Definition:

Branch-and-bound technique and dynamic programming are quite effective but their time-complexity often is too high and unacceptable for NP-complete tasks. it always finds the nearest local optimal of low quality. The goal of modern heuristics is to overcome this disadvantage. Hill-climbing algorithm is effective.

Step 0: Add domain-specific information to select the best path along which to continue searching

Step 1:. Define a heuristic function, h(n), that estimates the "goodness" of a node n. Specifically, h(n) = estimated cost (or distance) of minimal cost path from n to a goal state.

Step 2: The heuristic function is an estimate, based on domain-specific information that is computable from the current state description, of how close we are to a goal

Step 3 : Heuristic h(N)

Step 4: Evaluation function:

$f(N) = g(N) + h(N);$

where:

g(N) is the cost of the best path found so far to N

h(N) is an admissible heuristic

Step 5: Then, best-first search with this evaluation function is called A* search

Gossip Protocol

Gossip-based protocols have recently gained notable popularity. Apart from traditional applications for database replication gossiping algorithms have been applied to solve numerous other practical problems including failure detection , resource monitoring and data aggregation.

Advantage

- Continuous execution is possible.
- Suitable for heavy processes.
- Resource allocation handling is possible.
- No need for the global synchronization because the protocol gossip already capaplity for load balancing

Application

Web-based applications:

Gossip protocol for Dynamic Memory management in Large cloud Environment. The protocol is scalable, if any modification for the particular site means, the site owners arise the on demand to the site manager, the site manager create the request dispatcher to the resource manager, this application is fully based on online brokerage like. *Gossip-based protocols* are increasingly popular in large-scale distributed.... the various styles of *gossip*, such as aggregation and neighbor-set *management*, ... they each get a fair share of that node's communication and *memory* resources. ... create custom and flexible live *applications* or *web* pages

Conclusion

Gossip protocol that computes, in a distributed and continuous fashion, a heuristic solution to the resource allocation problem for a dynamically changing resource demand. The 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/applications.The results in this paper as building blocks for engineering a resource management solution for large-scale clouds.

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] "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.