A Dynamic Resource Dispatches in a Cloud Environment Using Join-Idle-Queue (JIQ)

V.Prasath
Assistant Professor
Department of CSE
PKIET, Karaikal
India

R.Buvanesvari
Assistant Professor
Department of CSE
PKIET, Karaikal
India

A.Shanmugapriya
UG Student
Department of CSE
PKIET, Karaikal
India

R.Annapoorani
UG Student
Department of CSE
PKIET,Karaikal
India

***Abstract*—In the cloud, the time for dispatching resources to the systems may be maximum due to the interaction occurs between the processors while allocating resources to them. We address this problem of dispatching resources in the cloud environment for the enhancement of scalability, adaptability and to increase its performance status. We propose a novel class of algorithm called Join-Idle-Queue for balancing the load among large systems which results in the efficient utilization of the processor in the cloud under its memory constraints. Our approach is mainly focused on dynamically distributing resources among systems at a very least time by using a middleware server. This algorithm continuously executes on dynamic, local input and does not require maximum time for allocating resources as other novel class of algorithms do. We evaluate our approach through simulation and experiment results demonstrate that this algorithm (JIQ) achieves good performance by consuming less time.**

***Keywords*—Cloud Computing,, Resource Dispatches, Join-Idle-Queue, Load Balancing.**

IJERT

I. INTRODUCTION

Cloud computing is a popular trend in current computing which attempts to provide cheap and easy access to computational resources. It provides a powerful computing model that allows users to access resources on-demand. Nowadays, cloud computing has become a key technology for online allotment of computing resources and online storage of user's data in a lower cost, where computing resources are available all the time, over the internet with pay per use concept. Cloud computing is business oriented concept where computing resources are outsourced by cloud provider to their client, who demand computing online [1]. Maintenance of cloud computing applications is also easier, because they do not need to be installed on each user's computer and can be accessed from different places. In the cloud, the time for dispatching resources to the systems may be maximum due to the interaction occurs between the processors while allocating resources to them. We take this problem into consideration and proposes a novel class of algorithm called Join-Idle-Queue for balancing the load among large systems which results in the efficient utilization of the processor in the cloud under its memory constraints .We are also focusing on load balancing of cloud computing with some JIQ techniques [2], which are responsible to manage the load when some node of the cloud system is overloaded and others are under loaded.

The aim of this paper is to demonstrate and discuss a critical role the load balancing of resources plays in improving the availability of resources and maintaining the performance in cloud systems at a very least time. In addition, JIQ technique adding capacity to the dynamic balance mechanism for the cloud. The experiment demonstrates that the algorithm is obtains better load-balancing degree and using less time in loading all tasks. It may result that the amount of computation assigned to each processor is balanced so that some processors do not sit idle while executing other tasks. In our approach, we present a centralized design that is the middleware server which provides the communication

Further the load between the clients and the other severs in the cloud balancing techniques, in the area of cloud computing, reduces costs associated with document management systems and also maximizes availability of resources [4]. This article discusses possible ways to improve the performance of cloud networks by the introduction of join-idle-queue load balancing technique that arranges the processor in a queue according to its memory space. When the client requesting for uploading their resources within the cloud, the defined middleware server will carry the request from the client and verify the join-idle-queue for the processor contained the memory needed to upload the resources.

It is used to dispatch jobs evenly to the front end servers. It uses the join-the-shortest-queue (JSQ) algorithm that dispatches jobs to the processor with the less time period. The JSQ algorithm is a greedy algorithm from the view of an incoming job, as the algorithm grants the job the highest instantaneous processing speed, assuming a processor sharing (PS) service discipline [2]. The JSQ algorithm is not optimal, but is shown to have great performance in comparison to algorithms with much higher complexity. Another benefit of the JSQ algorithm is its low communication overhead in traditional server farms, as the load balancer can easily track the number of jobs at each processor: all incoming requests connect through the centralized load balancer and all responses are also sent through the load balancer. The load balancer is hence aware of all arrivals of jobs to a particular processor, and all departures

As well, this makes tracking a simple task. No extra communication is required for the JSQ algorithm [2].

Our design goals ensures the following,

A. *Performance*

Ensure that performance is maintained at an acceptable level so that users don't experience significant lags when they are trying to carry out a particular task. The cloud should not be a performance detriment to the overall ability to do business. A traffic accelerator that optimizes the throughput between two sites and allows you to accelerate the data center-to-cloud resource combination is one good solution. The enormous flexibility of cloud hosting is one of its greatest benefits we consider computational and memory resources and the objective is to achieve max-min fairness among sites for computational resources under memory constraints and for allocating the resources in a minimal time. Under this objective, each site receives CPU resources proportional to its CPU demand.

B. *Adaptability*

The resource allocation process must dynamically and efficiently adapt to changes in the demand from users.

C. *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 sub linearly with both the number of machines and the number of sites.

D. *Minimum time consuming*

Since we are arranging the processors in a queue according to its memory, the time for dispatching load to the processors is minimum. It continuously executes on dynamic, local input and does not require maximum time for allocating resources.

These guiding principles performance, adaptability, scalability, minimum time consuming provide a useful checklist in helping ensure that your cloud performs exactly as you want it to.

However, this paper (a) dynamically adapts 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) [3].

II. ARCHITECTURE OF DISPATCHING RESOURCES

Our architecture associates with mainly dispatching and balancing the resources (load) in the cloud using Join-Idle-Queue technique. It has a middleware server responsible for dispatching the resources and a load balancer for allocating the resources to the processors based on the memory of each processors listed in the queue. A single hardware load balancer that accommodates hundreds of processors is both expensive and at times wasteful as it increases the granularity of scaling.fig.1.shows the Join-Idle-Queue (JIQ) architecture for large-scale load balancing with distributed dispatchers.

The central idea is to decouple discovery of lightly loaded servers from job assignment [7]. The basic version involves idle processors informing dispatchers at the time of their idleness, without interfering with job arrivals. This removes the load balancing work from the critical path of request processing. The challenge lies in the distributed nature of the dispatchers as the idle processors need to decide which dispatcher to inform what job to allocate. Informing a large number of dispatchers will increase the rate at which jobs arrive at idle processors, but runs the risk of inviting too many jobs to the same processor all at once and results in large queuing overhead [2].

The processor can conceivably remove itself from the dispatchers once it receives the first job, but this will require twice as much communication between processors and dispatchers. On the other hand, informing only one dispatcher will result in wasted cycles at idle processors and assignment of jobs to occupied processors instead, which adversely affects response times.

To solve the problem, the proposed JIQ algorithm load balances idle processors across dispatchers, which we call the secondary load balancing problem. In order to solve the primary load balancing problem of assigning jobs to processors, we first need to solve the secondary problem of assigning idle processors to dispatchers, which curiously takes place in the reverse direction. While the primary problem

IJERT

concerns the reduction of average queue length at each processor, the secondary problem concerns the availability of idle processors at each dispatcher.

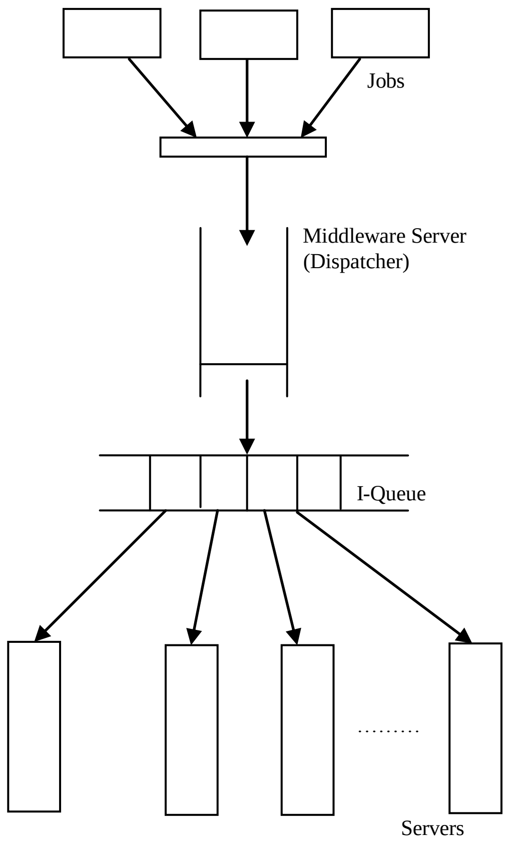

Fig.1.Architecture of JIQ

It is not a priori obvious that load balancing idle processors across dispatchers will outperform the algorithm because of the challenges outlined above. The analysis also applies to the FIFO service discipline [2]. Analyzing the performance of FIFO reentrant queues is outside the scope of this paper, but the analysis of FIFO queues is a necessary first step.

The main contributions of the paper are as follows:

1. We analyze and combine both the primary and secondary load balancing systems and find that the JIQ algorithm reduces the effective load on the system. The mean queue length in the system is shown to be insensitive to service time distributions for the processor sharing (PS) discipline in the large system limit.

2. The proposed JIQ algorithm incurs no communication overhead at job arrivals, hence does not increase actual response times. With equal or less complexity JIQ produces smaller queuing with the actual value depending on the load and processor-to-dispatcher ratio.

The evaluation of the performance of JIQ algorithm is based on simulation with a variety of service time distributions, corresponding to different application workloads.

The objective of the load balancing algorithm is to provide fast response time at each processor without incurring excessive communication overhead. In particular, communication overhead on the critical path, *i.e.*, at the arrival of a request, is to be avoided as it adds to the overall response time. Communication off the critical path is much less costly as it can ride on heartbeats sent from processors to job dispatchers signaling the health of the nodes.

III. ALGORITHM DESCRIPTION

The algorithm consists of the load balancing systems, which communicate through a data structure called I-queue. Together, they serve to decouple the discovery of idle servers from the process of job assignment. Fig.1. illustrates the overall system architecture with an I-queue of a dispatcher. An I-queue is a list of a subset of processors based on its memory capacity that have reported to be idle. All processors are accessible from each of the dispatchers. The load balancing system exploits the information of idle servers present in the I-queues, and avoids communication overhead from probing server loads. At a job arrival, the dispatcher consults its I-queue. If the I-queue is non-empty, the dispatcher removes the first idle processor from the I-queue and directs the job to this idle processor [2]. It refers to sharing of memory demand of the process while demand exceeds the capacity of each processor.

IJERT

When a processor becomes idle, it chooses one I-queue based on a load balancing algorithm and informs the I-queue of its idleness or joins the I-queue. For all algorithms in this class, each idle processor joins only one I-queue to avoid extra communication to withdraw from I-queues., or joins it. Depend upon the memory usage of the processors and demand of process, the resources are allocated to the processors by verifying the I-queue and thus the resources are dispatched only through middleware server [6].

To illustrate the effect of length of busy cycles on response times, compare the two busy cycle patterns on a single processor illustrated in Fig. 2. The letter ’b’ denotes ’busy’ and the letter ’i’ denotes ’idle’. The two patterns can result from different load balancing schemes in the system. The load is the same for both patterns, as they share the same mean idle time. However, pattern 2 indicates a much larger arrival rate than pattern 1 when the processor is idle. This results in shorter busy cycles and a much shorter response time.

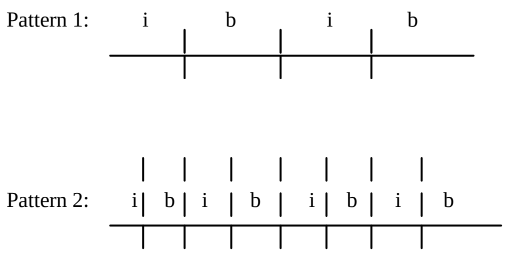

Fig.2. Busy (b) / idle (i) patterns of a processor.

This JIQ defines that the processors within the cloud are first arranges in the I-Queue based on the memory space available at each processors. Then the queue is loaded with the processors and the resource of high memory demand is allocated in high free memory processors and other resources are allocated in another processors corresponding to its memory demand. The most free memory processor in the cloud will receive the resource.

We study the performance of algorithms with analysis and simulation in the rest of the paper.

IV. EVALUATION THROUGH SIMULATION

We evaluate the class of JIQ algorithms against the shortest queue (SQ) algorithm for a variety of service time distributions via simulation. The deterministic distribution models applications with constant job sizes. We also evaluate the JIQ-SQ algorithms against the SQ algorithm, since the JIQ algorithms evaluated have strictly lower communication overhead than SQ [5]. Moreover, the overhead does not interfere with job arrivals, as in the case of SQ. It is based on Job-idle-queue algorithm and it refers to dispatching resources based on the memory demand of that resource. These resources will be allocated to the processors considering their available memory space. When these resources get allocated to the corresponding processor, it should be removed from the I-Queue. By re-considering the memory space available at the previously resource allocated processor, it can join to the I-Queue. The resource of high memory demand is allocated in high free memory processors and other resources are allocated in another corresponding processor. The most free memory processor in the cloud will receive the process. Thus the processor will join the I-Queue continuously after it has been removed based on its memory capacity, the time for allocating the incoming resources is minimum and also the performance get increased significantly.

V. CONCLUSION

In this paper we dynamically balancing load among the processor in the cloud Environment. The main aim of this paper is to reduce the time for allocating the resources to the processor. For balancing a load we introduce a technique called JIQ thus increases the performance speed of the processor and also it requires only a less time for resource allocation. We evaluate a simulation through this algorithm that we introduced is used to access all the processor so that the no processor in the cloud remain idle. In our approach the performance speed of the processor is increased by dynamically balancing the load among the processor in a minimal time.

VI. FUTURE WORK

The extension of the JIQ algorithms proves to be useful at very high load. It will be interesting to acquire a better understanding of the algorithm with a varying reporting threshold [2]. We would also like to understand better the relationship of the reporting frequency to response times, as well as an algorithm to further reduce the complexity of the JIQ-SQ algorithm while maintaining its superior performance.

IJERT

REFERENCES

[1] CloudComputing,accessed (23/01/2013),from http: //en.wikipedia.org/wiki /Cloud Computing overview with load balancing techniques.

[2] http://join-idle-queue algorithm.

[3] Fetahi Wuhib , Rolf Stadler, and Mike Spreitzer , [2] ‘‘A Gossip Protocol For Dynamic Resource Management in Large Cloud environment’’, IEEE Transaction on Network Service Management, vol.9 , no.2, in June 2012.

[4] Zenon Chaczko 1, Venkatesh Mahadevan 2, Shahrzad Aslanzadeh 1 and Christopher Mcdermid1 1, 3 & 4 University of Technology Sydney, Australia 2 Swinburne University of Technology, Australia, ‘‘Availability and Load Balancing in cloud Computing’’, International Conference on Computer and Software Modeling IPCSIT vol.14 (2011) © (2011) IACSIT Press, Singapore,2011.

[5] J. Name Stand. Abbrev., in press. V.Gupta, M. Harchol-Balter, K. Sigman, and W.Whitt ,” Analysis of join-the-shortest-Queue Routing for web server farms Perforance Evaluation” ,(64):1062,1081, 2007..

[6] Zhen Xiao ,Senior Member,IEEE ,Weijia Song and Qi Chen, ‘‘Dynamic Resource Allocation Using Virtual Machine for Cloud Computing Environment’’,IEEE on Parallel and Distributed System(TPDS),

[7] Ram Prasad Padhy P.Goutam Prasar Roa, “Load Balancing in Cloud Computing systems” in may 2011.