International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 15 (2015) pp 35253-35257

On Demand Replica Management of Application Servers with Service Level Agreements for Quality of Service Improvement

J.Parthasarathy

AP(S.G), Vivekanandha College of Technology for Women, Trichengodu, Tamil Nadu, India.

R.S.D. Wahidabanu

Principal, Government College of Engineering, Salem, Tamil Nadu, India.

ABSTRACT:
Service level agreements are the main issue of on demand computing, which hugely affects the quality of service. The SLA driven approaches has been discussed earlier to meet the agreements but suffers with the management of services and servers at higher load conditions. We propose two mechanisms, one for clustering application servers according to number of metrics like access frequency, server platform, response metrics and etc., another for dynamic generation of replica server at high load conditions. We discuss the load balancing of servers and services, to meet the service level agreements. The load balancing of service request is performed using a middleware which selects servers and redirects request to those servers. The middleware is capable of generating dynamic replica of server when it's necessary and will be removed at less load conditions.

Key Terms: Replica Server, SLA, QoS, Load Balancing.

1. INTRODUCTION:

The development of information technology opens the gate of on demand services, which could be accessed by the users as necessary and generated at demand. The services are provided according to SLA, and it becomes the response for the service provider to meet the agreements. The SLA consists of many criteria like: service availability, service reliability, and many more. Here the availability of service is the more dominant property, because the service has to be available at all the conditions for the user. The service provider or the QoS driven application has to meet the service availability and they have to plan about the required resources to meet the criteria.

Application server clustering is discussed in [1], where the servers are clustered like master and slave format and performs group communication. The problem with the solution is the server has to be up at all the time in all the nodes of the cluster to meet the requirements. This makes the resource unusable and reduces the throughput of the server which does not used at lower load conditions. So that the server has to be up and used at dynamic conditions when there is higher load arises.

Whenever the number of incoming http request increases the process of load balancing is comes into play. Every server has the bound in number of request handling and could not handle request more than that, also highly loaded server could not provide service at least response time. The load balancing procedure has to point all these issues before scheduling the request to a server.

The quality of service of any server or service is depending on throughput and timeliness, reliability. If the server response quickly then it will be good and will increase the throughput of the server. In order to increase the QoS, the server or service has to maintain the availability, reliability metrics at all levels of service agreements.

Replica is a method of generating the copy of service or server which could be placed at any node and could start servicing by runtime. This has to be done at runtime and managed based on load conditions. we propose such a framework to create replica of a server and start servicing at runtime with dynamic conditions. The newly generated server will be clustered into the early clusters according to the clustering parameters.

2. BACKGROUND:

A System for Dynamic Server Allocation in Application Server Clusters [1], investigates a system for allocating server resources to applications dynamically, thus allowing applications to automatically adapt to variable workloads. Such a scheme requires meticulous system monitoring, a method for switching application servers between text it {server pools} and a means of calculating when a server switch should be made (balancing switching cost against perceived benefits). Experimentation is performed using such a switching system on a Web application test bed hosting two applications across eight application servers. The test bed is used to compare several theoretically derived switching policies under a variety of workloads. Recommendations are made as to the suitability of different policies under different workload conditions.

Construction and Application of Linux Virtual Server Cluster for Scientific Computing[2], describe a cluster server, Linux virtual server cluster, formed by various inexpensive personal computers for scientific computing. This cluster has many favorable features, including lower cost, load balancing, and dynamic scalability.

Enhancing an Application Server to Support Available Components [3], describes how replication for availability can be incorporated within the middle and back-end tiers, meeting all these challenges. This paper develops an approach that requires enhancements to the middle tier only f

or supporting replication of both the middleware back-end tiers. The design, implementation, and performance evaluation of such a middle-tier-based replication scheme for multi database transactions on a widely deployed open source application server (JBoss) are presented.

Dynamic load balancing algorithm for scalable heterogeneous web server cluster with content awareness [4], propose a DLB algorithm for scalable heterogeneous server cluster using content awareness. The algorithm considers server's processing capability, queue length, utilization ratio etc. as load indices. As the cluster supports multiple services, at the primary level, we have used content awareness forwarding algorithm and at the secondary level, waited round robin algorithm has been used.

Multi-Cloud Deployment of Computing Clusters for Loosely-Coupled MTC Applications [5], explore this scenario to deploy a computing cluster on top of a multi-cloud infrastructure, for solving loosely-coupled Many-Task Computing (MTC) applications. In this way, the cluster nodes can be provisioned with resources from different clouds to improve the cost-effectiveness of the deployment, or to implement high-availability strategies. We prove the viability of this kind of solutions by evaluating the scalability, performance, and cost of different configurations of a Sun Grid Engine cluster, deployed on a multi-cloud infrastructure spanning a local data-center and three different cloud sites: Amazon EC2 Europe, Amazon EC2 USA, and Elastic Hosts. Although the test bed deployed in this work is limited to a reduced number of computing resources.

Symphony: A Scheduler for Client-Server Applications on Coprocessor-Based Heterogeneous Clusters [12], propose a novel scheduler called Symphony that enables efficient, dynamic sharing of a GPU-based heterogeneous cluster across multiple concurrently-executing client-server applications, each with arbitrary load spikes. Symphony performs three key tasks: it (i) monitors the load on each application, (ii) collects past performance data and dynamically builds simple performance models of available processing resources and (iii) computes a priority for pending requests based on the above parameters and the requests' slack. Based on this, it reorders client requests across different applications to achieve acceptable response times.

Improving Application Placement for Cluster-Based Web Applications [13], first propose and define new optimization objectives: limiting the worst case of each individual server's utilization, formulated by a min-max problem. A novel framework based on binary search is proposed to detect an optimal load balancing solution. Second, we define system cost as the weighted combination of both placement change and inter-application communication cost. By maximizing the number of instances of dependent applications that reside in the same set of servers, the basic load-shifting and placement-change procedures are enhanced to minimize whole system cost.

DynaPlan: Resource placement for application-level clustering [14], a method that improves the quality of failover planning by allowing the expression of a wide and extensible range of considerations, such as multidimensional resource consumption and availability, architectural compatibility, security constraints, location constraints, and policy considerations, such as energy-favoring versus performance-favoring.

A System for Dynamic Server Allocation in Application Server Clusters [15], This paper investigates a system for allocating server resources to applications dynamically, thus allowing applications to automatically adapt to variable workloads. Such a scheme requires meticulous system monitoring, a method for switching application servers between server pools and a means of calculating when a server switch should be made (balancing switching cost against perceived benefits).

3. PROPOSED METHOD:

The proposed method has three stages namely: middleware service, load balancing, and clustering. The middleware performs the service selection and server selection and controls the load balancing and clustering. The load balancing is performed when there are unbalanced job load present in different servers of the clusters. The clustering is performed to group set of servers according to their characteristics.

3.1 Middleware Service:

The middleware service is the request handler in the proposed model which controls the overall flow of the framework. whenever the middleware receives a request from the user, it selects one of server by which the request has to be handled. The selection of server is based on the service agreements and quality of service parameters and access frequency, and response time. If there is no server is free to handle the request then the middleware generates a replica of a server and run it on a machine then handover the request to the server.

Algorithm:

step1: start
step2: initialize service history Sh, available machines M, clusers C.
step3: Identify Request HttpRq = Ø(Request).
step4: Identify the server and platform of request HttpRq.
Srq = $\int_1^N HttpRq \in (C \times M)$.
step5: compute load conditions of Srq.
Ld = $\sum Cr{\cdot}q \in (Sr{\cdot}q)$.
step6: select most least weighted server from Ld.
step7: Redirect request to selected server.
step8: stop.

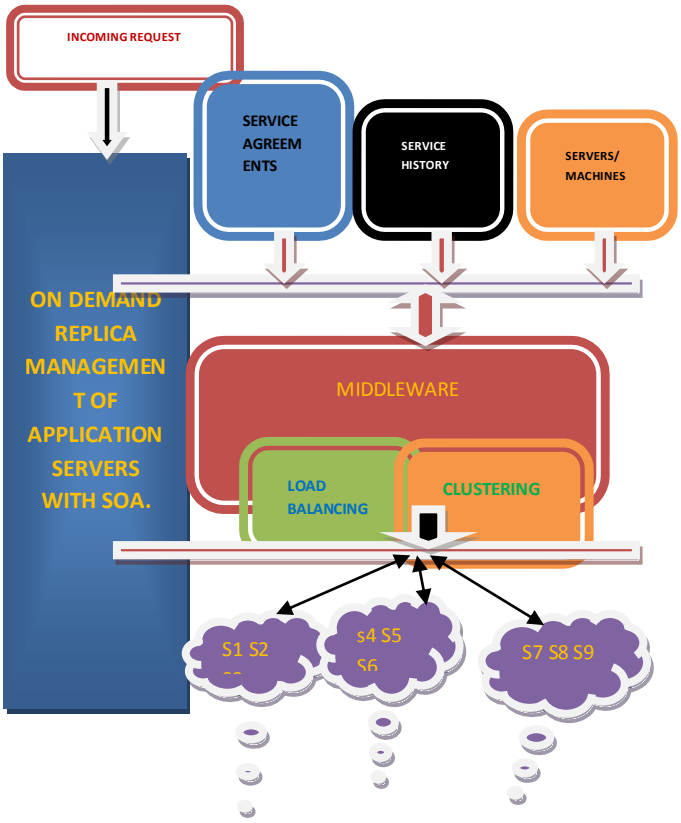


Figure1: Proposed Method Architecture.

3.2 Load Balancing:

Load balancing is performed at regular interval and at each time whenever the middleware receives the request. From set of servers S available and clusters C, we perform load balancing based on the following metrics like timeliness, availability, response time, access frequency and request flow.

Algorithm:

step1: start
step2: initialize request flow Rf, servers S, Clusters C, Access Frequency Af.
step3: for each cluster C_i from C
for each Server S_i from C_i
compute access frequency Af = $\sum_1^x Alog(S_i) \in (C_i)$
$Af(S_i)$ = Af.
compute average response time Rt = $\sum_1^x (Alog(Si) \times Ts) - (Alog(si) \times Tr))/x$
Ts- Time submitted
Tr – Time Replied
end
step4: Receive request Rq.
Identify server with least Af value and High Response time.
S = Min(Af)×Max(Rt).
step5: if((Af> Ath) and Rt> Rth)
generate Replica.
Handover request at new server.
cluster server into the available cluster.
else
Handover request to selected server S.
step6: stop.

3.3 Clustering:

Clustering of servers is performed based on the characteristics of servers and their platform of nature and kind of services it provides. For example set of servers working under Linux platform are grouped and others working under other operating systems are grouped separately. Also according to the service metrics like the capacity of server as number of request it can handles at runtime is used to group the servers. The servers with less capacity are grouped separately and high capacity servers are grouped separately.

Algorithm:

step1: start
step2: initialize clusters C.
step3: read number of available servers S.
step4: read server characteristics capacity cap, platform pf.
step5: Assign random labels to the servers.
step6: perform K-means clustering.
step7: stop.

3.4 Replica Management:

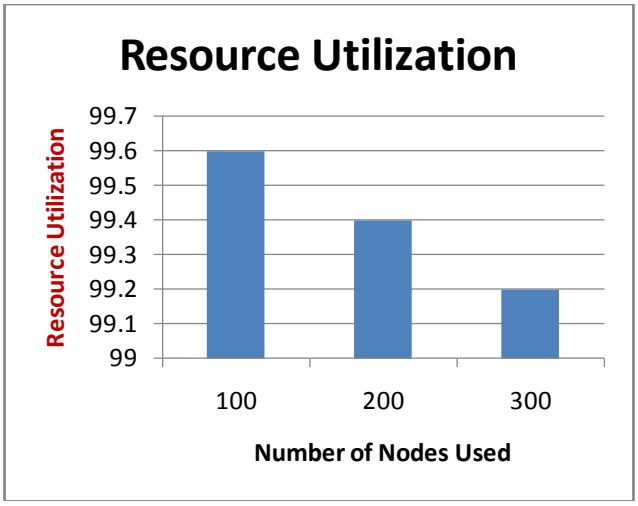
The server replica is generated when there is huge load present in all the servers and overall response time of the servers crosses a threshold. The middleware generates a server replica and start servicing in new node and handover the request to the newly started server. Newly started server will be clustered into the available clusters.

4. RESULTS AND DISCUSSION:

The proposed approach has been implemented in different operating systems and the performance of the approach has been evaluated with 300 nodes as servers. The initial clustering is performed with 90 percent of nodes ie 270 nodes and grouped as 5 clusters. we have used servers with following characteristics as mentioned in the table 1.

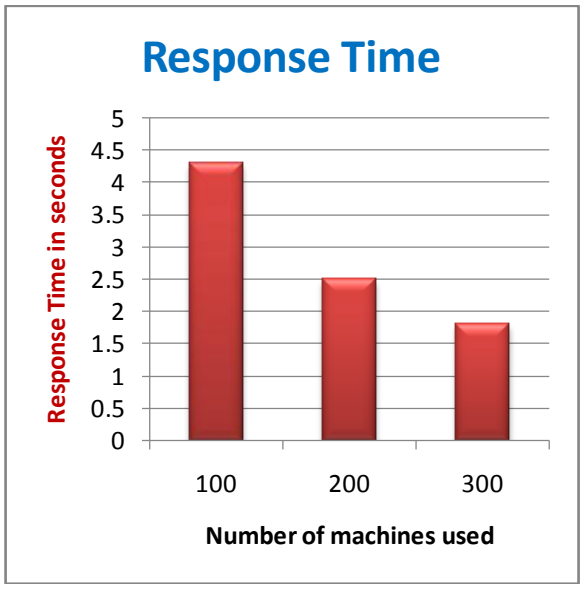
Table1: configurations used for clustering.

Cluster	Capacity	Operating System	Server
1	3000	Linux	JBoss
2	2500	Windows	Apache
3	2000	MacOs	JBoss
4	1500	Windows	Apache
5	1000	Windows	Glassfish



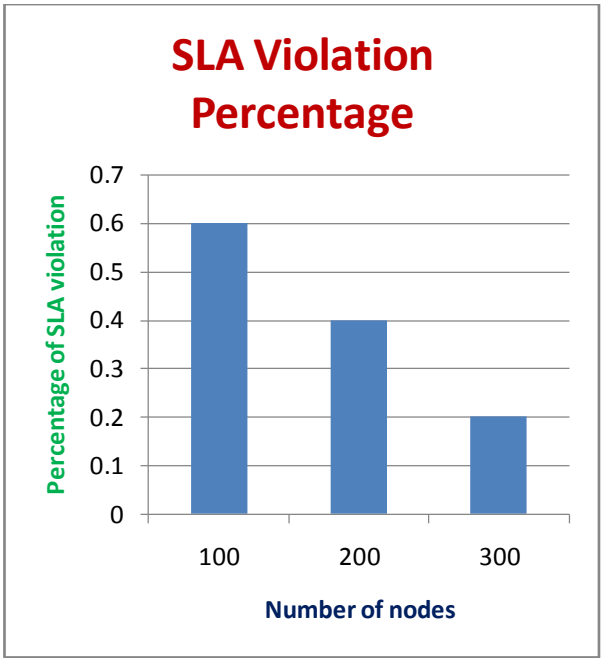
Graph1: shows the resource utilization produced.

The graph1 shows the resource utilization achieved by the proposed method. It is clear that the proposed method has achieved higher utilization at all level of density.



Graph2: Response Time of proposed method.

The graph2 shows the response time achieved by the proposed method. It is clear that the proposed real-time replica management approach has reduced the response time with different number of nodes.



Graph3: shows the percentage of SLA violation.

The graph 3 show the violation generated by the proposed method according to SLA. It shows that the proposed method has violated very negligible level and does not affect the QoS at any stage.

5. CONCLUSION:

We propose a new on demand replica management framework, which clusters application servers according to access frequency, response time and capacity of servers. The load balancing is performed based on the access frequency and response time, whenever the access threshold and response threshold crosses a limit the middleware generates a new copy of required server and handovers the request to the server. The new replica will be clustered into the available servers. The proposed approach has produced higher results with increased performance and increases throughput of the overall framework.

REFERENCES

[1] Chester A.P, A System for Dynamic Server Allocation in Application Server Clusters, Parallel and Distributed Processing with Applications, pp:130-139, 2008.

[2] Hong Tong, Construction and Application of Linux Virtual Server Cluster for Scientific Computing, IFIP, Network and Parallel Computing, pp: 287-289, 2008.

[3] Kistijantoro, Enhancing an Application Server to Support Available Components, IEEE Transactions on Software Engineering, vol:34, Issue:4, pp:531-545, 2008.

[4] Tiwari A, Dynamic load balancing algorithm for scalable heterogeneous web server cluster with content awareness, Trends in Information science and computing, pp:143-148, 2010.

[5] Rafael Moreno-Vozmediano, Multi-Cloud Deployment of Computing Clusters for Loosely-Coupled MTC Applications, IEEE Transaction on Parallel and distributed systems, 2010.

[6] E. Huedo, R. Montero, I. Llorente, A Modular Meta-Scheduling Architecture for Interfacing with Pre-WS and WS Grid Resource Management Services, Future Generation Computer Systems 23 (2): 252–261, 2007.

[7] B. Sotomayor, R.S. Montero, I.M. Llorente, I. Foster, Virtual Infrastructure Management in Private and Hybrid Clouds, IEEE Internet Computing 13(5): 14–22, 2009.

[8] M. Tsugawa, J. Fortes, A Virtual Network (ViNe) Architecture for Grid Computing, in: In Proc. IEEE Intl. Parallel and Distributed Processing Symposium (IPDPS06), 2006, pp. 1–10.

[9] M. Matos, A. Sousa, J. Pereira, R. Oliveira, CLON: Overlay Network for Clouds, in: Proceedings of the Third Workshop on Dependable Distributed Data Management, 2009, pp. 14–17.

[10] G. Juve, E. Deelman, K. Vahi, B. P. Berman, B. Berriman, and P. Maechling, Scientific Workflow Applications on Amazon EC2. Workshop on Cloud-based Services and Applications in conjunction with 5th IEEE Int. Conference on e-Science, 2009.

[11] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, The cost of doing science on the cloud: the Montage example. Proceedings of the 2008 ACM/IEEE conference on Supercomputing, 2008.

[12] Rafique M.M, Symphony: A Scheduler for Client-Server Applications on Coprocessor-Based Heterogeneous Clusters, cluster Computing, pp:353-362, 2011.

[13] Chein Tyan, Improving Application Placement for Cluster-Based Web Applications, IEEE transaction on network and service management, Vol:8, issue:2, pp:104-115, 2011.

[14] Harper R.E, DynaPlan: Resource placement for application-level clustering., IEEE conference on Dependable Systems and Networks Workshops (DSN-W), pp:271-277, 2011.

[15] Chester, A.P., Xue, J.W.J., He, L. and Jarvis, S.A. (2008) A System for Dynamic Server Allocation in Application Server Clusters. In: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA'08), 10-12 December, 2011, Sydney, Australia.

[16] D.B. Ingham, S.K. Shrivastava, and F. Panzieri, “Constructing Dependable Web Services, ” IEEE Internet Computing, vol. 41, no. 1, Jan.-Feb. 2000.