ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 5, Issue 10, October 2016

Improvised Provision for Load Balancing in Content Delivery Networks

Chetana B. Bhagat, Dilip K. Budhwant

Abstract— **This paper presents how four load balancing algorithms affect the total execution time of the parallel job using JPPF (Java Parallel Processing Framework). JPPF is Open Source tool that enables applications with large processing power requirements to be run on any number of nodes, in order to reduce their processing time . The algorithms differ in the way of balancing scheduling tasks between the nodes. One of the algorithms uses the fixed (static) number of tasks given by a user. The other three algorithms use some form of the adaptive load balancing. There are two sets of the measurements presented using these four algorithms for executing a parallel job of matrix multiplication. The results of these measurements led to the conclusion that the static load balancing algorithms spend less time on communication between server and a nodes compared to the other.**

Index Terms— **CDN, Load balancing, JPPF, auto tuned algorithm, proportional algorithm, manual algorithm, ICLBS.**

I. Introduction

Recently, Java has emerged as a language of choice for parallel programming. As one of the open source tools, Java arouse a lot of interest among professional software engineers. By using the JPPF, they can use things that enable easier parallel programming [3]. One of the most helpful tools are the load balancing algorithms. Usually, other frameworks that support parallel programming use only one algorithm to schedule jobs. JPPF offers four load balancing algorithms which give users a possibilities to analyze their effect on the total execution time of parallelized program. This will be explained in the following sections.

Algorithms Used For Load Balancing

In the following, we will describe the most common algorithms used for the purpose of load balancing in a CDN. Such algorithms will be considered as benchmarks for the evaluation of the solution we have presented in this paper.

The simplest static algorithm is the *Random b*alancing mechanism (RAND). In such a policy, the incoming requests are mainly distributed to the servers in the network with a uniform probability.

The *Least-Loaded* algorithm (LL) is a renowned dynamic strategy for load balancing. It assigns the incoming client request to the currently least loaded server. Such kind of approach is adopted in several commercial solutions. Unfortunately, it tends to rapidly saturate the least loaded server until the new message is propagated [12]. Alternative solutions can rely on *Response Time* to select the server: The request is allocated to the server that shows the fastest response time [13].

The *Two Random Choices* algorithm [14] (2RC) randomly chooses two servers and assigns the request to the least loaded one between them.

We propose a highly dynamic distributed strategy based on the periodical exchange of information about the status of the nodes in terms of load. By exploiting the multiple redirection mechanism offered by HTTP, our algorithm tries to achieve the global balancing through a local request redistribution process. Upon arrival of a new request, indeed, the CDN server can either elaborate locally the request or redirect it to other servers according to a certain decision rule, which is based on the state information exchanged by the servers. Such an approach limits state exchanging overhead to just local servers. .

II. Literature survey

Request routing in a CDN is usually concerned with the issue of properly distributing client requests in order to achieve load balancing among the servers involved in the distribution network. Several mechanisms have been proposed in the literature. They can usually be classified as either *static* or *dynamic*, depending on the policy adopted for server selection [15]. Static algorithms select a server without relying on any information about the status of the system at decision time. Static algorithms do not need any data retrieval mechanism in the system, which means no communication overhead is introduced. These algorithms definitely represent the fastest solution since they do not adopt any sophisticated selection process. However, they are not able to effectively face anomalous events like flash crowds.

Dynamic load-balancing strategies represent a valid alternative to static algorithms. Such approaches make use of information coming either from the network or from the servers in order to improve the request assignment process. The selection of the appropriate server is done through a

Manuscript received November, 2016.

***Chetana B. Bhagat**,, CSE Department ,Dr.Babasaheb Ambedkar Marathwada University/ Jawaharlal Nehru Engineering College,Aurangabad, Aurangabad,,India,8275452735,7040212787.*

***Dilip K. Budhwant**, CSE Department ,Dr.Babasaheb Ambedkar Marathwada University/ Jawaharlal Nehru Engineering College,Aurangabad.,Aurangabad,,India,9423148306.*

collection and subsequent analysis of several parameters extracted from the network elements. Hence, a data exchange process among the servers is needed, which unavoidably incurs in a communication overhead.

The redirection mechanisms can be implemented either in a *centralized* or in a *distributed* way [15]. In the former, a centralized element, usually called *dispatcher*, intercepts all the requests generated into a well-known domain, for example an autonomous system, and redirects them to the appropriate server into the network by means of either a static or a dynamic algorithm. Such an approach is usually adopted by commercial CDN solutions. With a distributed redirection mechanism, instead any server receiving a request can either serve it or redistribute it to another server based on an appropriate (static or dynamic) load-balancing solution.

Depending on how the scheduler interacts with the other components of the node, it is possible to classify the balancing algorithms in three fundamental models. [16]
In a rate-adjustment model, instead the scheduler is located just before the local queue: Upon arrival of a new request, the scheduler decides whether to assign it to the local queue or send it to a remote server. Once a request is assigned to a local queue, no remote rescheduling is allowed. Such a strategy usually balances the request rate arriving at every node independently from the current state of the queue. No periodical information exchange, indeed, is requested. In a hybrid-adjustment strategy for load balancing, the scheduler is allowed to control both the incoming request rate at a node and the local queue length. Such an approach allows to have a more efficient load balancing in a very dynamic scenario, but at the same time it requires a more complex algorithm.

In the context of a hybrid-adjustment mechanism, the queue-adjustment and the rate-adjustment might be considered respectively as a fine-grained and a coarse-grained process. Both centralized and distributed solutions present pros and cons depending on the considered scenario and the specific performance parameters evaluated. As stated in [17], although in some cases the centralized solution achieves lower response time, a fully distributed mechanism is much more scalable. It is also robust in case of dispatcher fault, as well as easier to implement.

Finally, it imposes much lower computational and communication overhead.

III. PERFORMANCE MEASUREMENT SCENARIOS

To determine how the load balancing algorithms effect on the total execution time of parallel program, execution time measurement was conducted in four scenarios. [11] computer with following characteristics were used:

- CPU, Intel Core 2 Duo E7200.
- clock frequency, 2.533 GHz.
- RAM size. 3 GB (visible to OS),
- Operating System. Windows 7 Professional
- Java Run Environment (JRE) 7

One computer was used as a server, while the other as nodes. Using the instance of administration console on server, ten nodes were administrated.

TABLE I
SETTINGS FOR PROPERTIES OF 4 LOAD BALANCING ALGORITHMS

LOAD BALANCING	PARAMETERS
MANUAL	SIZE = 5

PROPORTIONAL	PERFORMANCECACHESIZE=2000
	PROPORTIONALITYFACTOR = 2

ICLBS	PERFORMANCECACHESIZE = 2000
	MAXACTIONRANGE = 10
	PERFORMANCEVARIATIONTHRESHOLD = 0.001

Parameters of these load balancing algorithms are explained below :

1) **Manual algorithm** - divides the job on a fixed number of tasks. This algorithm has only one parameter size – fixed number of tasks per node.

2) **Proportional algorithm** - is purely adaptive [5] and based solely on the known previous performance of the nodes. Each bundle size is determined in proportion to the mean execution time to the power of N. Here, N is one of the algorithm's parameters, called "proportionality factor". The mean time is computed as a moving average over the last M executed tasks for a given node. M is the other algorithm parameter, called "performance cache size". Bundle size for each node depends on others. Implementation of this algorithm is based on this formula :

n = current number of nodes attached to the driver max = maximum number of tasks in a job in the current queue state mean i = mean execution time for node i. S_i = number of tasks to send to node i. p = proportionality factor parameter.

Then it defines:

$$S = \sum(\frac{1}{mean})^P$$

S_i is then calculated as

$$S = max * \frac{(\frac{1}{mean})^P}{S}$$

The bundle size for each node is proportional to its contribution to sum S, hence the name of the algorithm. Every time performance data is fed back to the server, all node bundle sizes are re-computed.
This algorithm has two parameters :
a) performanceCacheSize - the maximum size of the performance samples cache.
b) proportionalityFactor.

4) **ICLBS** - The implemented algorithm consists of two independent parts: a procedure that is in charge of updating the status of the neighbors' load, and a mechanism representing the core of the algorithm, which is in charge of distributing requests to a node's neighbors.

The pseudocode of the algorithm is:

```
//peer status update
prob_space[0] = 0; load_diff_sum = 0;
for ( j=1; j<=n; j++ ){
   if ( load_i < peer [j].load){
   //assign to load_i
   }
   else
       {
        //assign to peer[j].load
       }
//normalize the vector element
Update_prob_space (load_diff_sum, prob_space);
}

//balancing process
if (prob_space[] == NULL) //no neighbors
serve_request ();
else{
int x = rand () // random number generation 0 or 1
while ( queue.hasElement){
if ( x == 1) {
send_to ( right_node )
}
else
    {
     Send_to ( left_node )
     }
  }
}
```

We have demonstrated the effectiveness of such a discrete version with the help of simulations. Let $q_i(t)$ be the queue occupancy of server i at time t. We consider the instant arrival rate $\alpha_i(t)$ and the instant service rate $\delta_i(t)$. The CDN servers' queues is given by

$$\frac{dq_i(t)}{dt} = q_i(t) = \alpha_i(t) - \delta_i(t) \quad (1)$$

for $i = 1$……N.

Equation (1) represents the queue dynamics over time. In particular, if the arrival rate is lower than the service rate, we observe a decrease in queue length..

IV. EXPERIMENTAL RESULTS

- time needed to schedule data to node
- time for executing operations on node and
- time it takes to return the results to client

TABLE III
– time needed to schedule data to nodes,
LOCAL DISTRIBUTION PROCESS (FOR 0.5 SEC)

AVG ARRIVAL RATE (S)	AVG SERVICE RATE (S)	REDIRECTION
3.653	13.76	0%

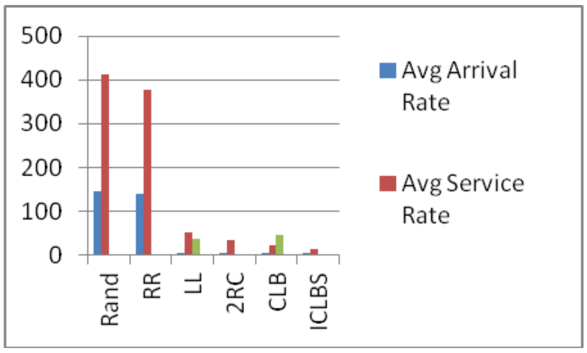

Fig.1 Graphical Representation of Performance in Local Distribution Process for 0.5s

TABLE IIIII
LOCAL DISTRIBUTION PROCESS (FOR 1 SEC)

AVG ARRIVAL RATE (S)	AVG SERVICE RATE (S)	REDIRECTION
2.117	5.86	0%

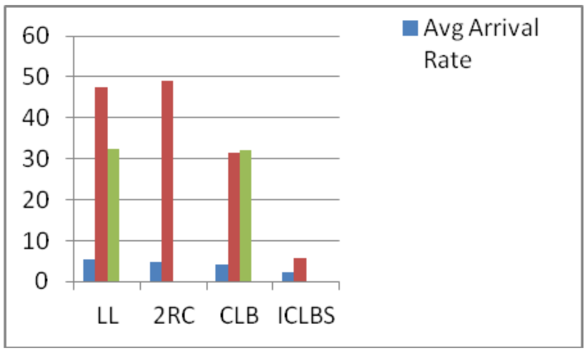

Fig.2 Graphical Representation of Performance in Local Distribution Process for 1s

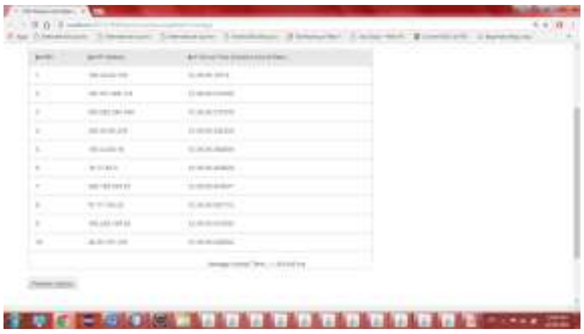

Fig.3 Flash Crowd Generation

TABLE IVV
FLASH CROWD PROCESS (FOR 1 SEC)

AVG ARRIVAL RATE (S)	AVG SERVICE RATE (S)	REDIRECTION
127.148	10.42	30%

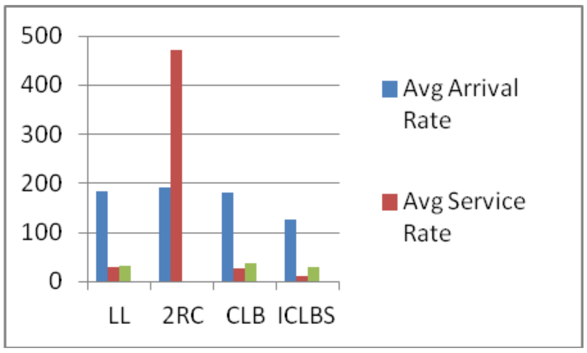

Fig.4 Graphical Representation of Performance in Flash Crowd Process for 1s

In other algorithms, the network communication comes into play because they are using heuristics for finding the optimal deployment of tasks. This leads to increased total execution time of parallel program..

V. CONCLUSION

The above comparison shows that static load balancing algorithms are more efficient compared to adaptive ones and it is also ease to predict the behaviour of static algorithms. Adaptive algorithms using the variable granularity of work creates more network communication between servers and nodes. Increased network communication leads to increased total execution time of parallel program, and poor granularity of work can also significantly increase the total executing time.

ACKNOWLEDGMENT

First and foremost, I would like to thank my guide, for unconditional guidance and support. I will forever remain grateful for the constant support and guidance extended by guide, in making this paper. Through our many discussions, he helped me to form and solidify ideas. The invaluable discussions I had with him, the penetrating questions he has put to me and the constant motivation, has all led to the development of this project.

REFERENCES

[1] Matthew L., Content Delivery Network (CDN) 2A Reference Guide, detail, April 10, 2014, http://Informationweek.com/detail/RES/9959901 81212.html.
[2] OLNN.CN.CDN, The Content distributed network technology, server, May 28, 2015, http://olnn.cn/html/server/6/20070528/3660.html
[3] W. D. Zhao, Content routing algorithms achieving network proximity in content delivery networks, Journal of Zhejiang University, Vol. 38, No. 4, p.414 -419, April 2014.
[4] (2011) The JPPF website [Online]. Available: http://jppf.org/about.php
[5] (2011) The Wikipedia website [Online]. Available:http://en.wikipedia.org/wiki/Load_balancing(computing)
[6] (2011) The JPPF Performance website [Online]. Available: http://jppf.org/doc/v2/index.php?title=JPPF_Performance
[7] (2005) Reinforcement Learning: An Introduction - Richard S. Sutton and Andrew G. Barto (Book)
[8] Derek L. Eager, Edward D. Lazowska , John Zahorjan, "Adaptive load sharing in homogeneous distributed systems", IEEE Transactions on Software Engineering, v.12 n.5, p.662-675, May 1986
[9] Monte Carlo method [Online]. Available :http://en.wikipedia.org/wiki/Monte_Carlo_method
[10] M. Dahlin, "Interpreting stale load information," IEEE Trans. Parallel Distrib. Syst., vol. 11, no. 10, pp. 1033–1047, Oct. 2000.
[11] R. L. Carter and M. E. Crovella, "Server selection using dynamic path characterization in wide-area networks," in Proc. IEEE INFOCOM, Apr. 1997, vol. 3, pp. 1014–1021.
[12] M. D. Mitzenmacher, "The power of two choices in randomized load balancing," IEEE Trans. Parallel Distrib. Syst., vol. 12, no. 10, pp.1094–1104, Oct. 2001.
[13] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu, "The state of the art in locally distributedWeb-server systems," Comput. Surveys,vol. 34, no. 2, pp. 263–311, Jun. 2002.
[14] Z. Zeng and B. Veeravalli, "Design and performance evaluation of queue-and-rate-adjustment dynamic load balancing policies for distributed networks," IEEE Trans. Comput., vol. 55, no. 11, pp. 1410–1422, Nov. 2006.
[15] V. Cardellini, M. Colajanni, and P. S. Yu, "Request redirection algorithms for distributed Web systems," IEEE Trans. Parallel Distrib. Syst., vol. 14, no. 4, pp. 355–368, Apr. 2003.

Chetana B. Bhagat, CSE Dept., Dr. Babasaheb Ambedkar Marathwada University/ Jawaharlal Nehru Engineering College, Aurangabad,India, 8275452735,7040212787.

Dilip K. Budhwant, CSE Dept., Dr. Babasaheb Ambedkar Mqrathwada University/ Jawaharlal Nehru Engineering College, Aurangabad, India, 9423148306.