Approach for Processor to Dispatcher Load Balancing in Distributed Networks

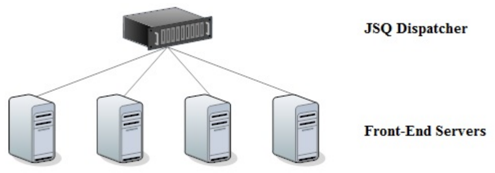
KULDEEP SHARMA
Maharaja Agrasen University
and
DEEPAK GARG
Thapar University

Load balancing is extensively used in distributed network applications to decrease the response times. In this paper, the focus is on backend load balancing (processor-to-dispatcher). Many algorithms are devised for front end load balancing e.g. JSQ, SQ(d) and JIQ. Join-Idle-Queue(JIQ) goes well in a distributed environment like cloud computing. In JIQ approach, at the backend, a processor joins the queue on either random or sampled basis. In both the cases, I-Queue of any dispatcher might remain empty resulting in degrading the performance. After finishing the job, the processor should join the dispatcher whose I-Queue is empty. To achieve this, we have used a dequeue to track the dispatcher with empty I-Queue. As the processor finishes the current job and reaches the idle state, it should refer the dequeue and join the dispatcher whose I-Queue is empty.

Keywords: Load Balancing, Distributed Networks, Secondary Load Balancing, Backend Load Balacing

1. INTRODUCTION

Load balancing is an official technique for the allocation of resources among different jobs. In standard web server farm, a concentrated hardware load balancer is utilized to send jobs equally to the front end servers [Chou and Abraham 1982]. The most famous and result oriented algorithms are Join-the-shortest-queue (JSQ) algorithm, SQ(d) algorithm and Join-Idle-Queue (JIQ) algorithm. JSQ is the most prevalent technique and used in processor sharing server farms like in CISCO local Director, F5 Application Delivery Controller, Microsoft SharePoint and IBM Network Dispatcher [Gupta et al. 2007] [Wang and Morris 1985]. In JSQ, a new request is allocated to the server which has the least count of unprocessed requests. In this manner, JSQ endeavors to adjust stack over the servers, diminishing the probability of a server having a few jobs whereas another servers are idle. From the aspect of incoming request, it is a greedy strategy for the instance of PS servers because the incoming request would have a preference to share a server with as small number of jobs as possible [Squillante and Nelson 1991]. In JSQ algorithm, all incoming job requests come through a centralized load balancer and similar is true for the responses. The load balancer is now conscious of all the arrival and departure requests to an individual front end server, making it simple with no extra communication required for tracking. A customary web server ranch contains just a couple of servers whereas appropriated server farms have hundreds or a great many processors for the front end alone. The capacity to scale horizontally in and out to adapt to the elasticity of demand is highly valued in data centers. A solitary equipment load balancer that accommodates many processors are both expensive and inefficient as it increases the granularity of scaling [Bramson et al. 2010]. Later, need for distributed dispatcher was felt in which centralized load balancing did not prove to be efficient enough [Applewhite et al. 1982]. Every server has its own list of arrivals and departures, so the actual load of the system could not be determined. To overcome this, it was required to have a communication channel which in turn, increased the overhead of communication and the load [Vvedenskaya et al. 1996]. $\lambda(n)$

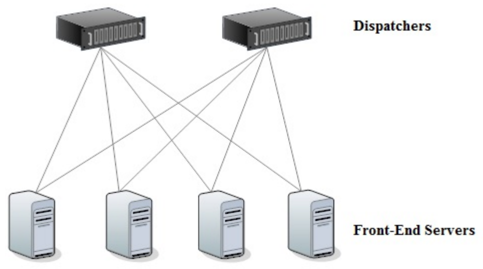


. Figure. 1: Join-the-shortest-queue (JSQ) algorithm

denotes the conditional arrival rate into the first queue, Specifically, we define:

$$\lambda(n) = \lim_{t\to\infty} \frac{A_n(t)}{T_n(t)}$$

where $A_n(t)$ is the count of arrivals amid the time interval $[0, t]$ finding n occupations in the queue 1, where $T_n(t)$ is the aggregate time amid $[0, t]$ during which there are n occupations in the queue 1. Every dispatcher autonomously adjusts its jobs since just a small amount of jobs use a specific dispatcher. The dispatcher has no information of the present number of jobs in every server which makes the execution of the JSQ algorithm troublesome.



. Figure. 2: Distributed Dispatchers

1.1 Motivation

(1) A dispatcher receives a job for processing but its I-Queue is empty since no processor is available in idle state. Due to this, job will be assigned to another server on random basis. It will send request but this process will degrade the performance and moreover we have idle processors in other I-Queues which are underutilized.

(2) When the I-Queue is empty, the request is sent to another server randomly but on the other hand, this server is associated with the I-Queue of other dispatcher which shows its (server) state as idle, whereas it is not. The server is already processing the job and will be in idle state once the job is accomplished.

(3) After completing the job, the processor should join the dispatcher with an empty I-Queue.

1.2 Our Contribution

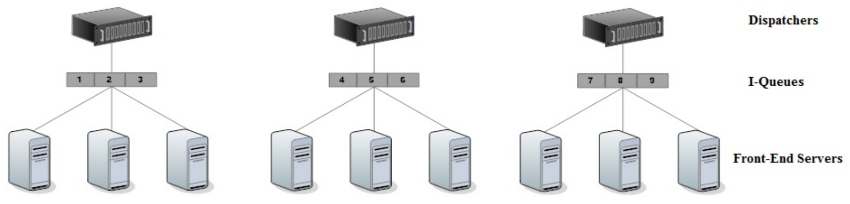
To overcome the limitations of backend load balancing in JIQ, our approach introduces dequeue to be associated in the secondary load balancing of JIQ. When I-Queue of any dispatcher is exhausted, its entry is recorded in the dequeue. So as soon as the server finishes the job, it will refer this dequeue to be associated with the dispatcher (whose I-Queue is empty). All the limitations (as said above) are due to the emptiness of I-Queue. So dequeue can be effectively used here to ensure the availability of server in every I-Queue. We have considered JIQ as fundamental part for our proposed approach. Our commitment is towards the improvement of backend load balancing(secondary load balancing) by guaranteeing the accessibility of server in every I-Queue.

1.3 Organization of paper

In this paper, we have proposed the exit plan for backend load balancing which is also referred as secondary load balancing (processor-to-dispatcher). Section 2 covers the Join-Idle-Queue algorithm [Lu et al. 2011]. Section 3 presents the system model. In section 4, the proposed approach is discussed. Section 5 embodies analysis. We have concluded the paper in section 6.

2. RELEVANT WORK

Join-Idle-Queue algorithm is suitable for the distributed environment where various dispatchers are put. In this approach, I-Queue is maintained by each dispatcher in which related server's entrance is kept. As an incoming request lands at the dispatcher, it sends to the processor referring the I-Queue and erase the entry of the processor from the I-Queue. If the I-Queue is unfilled then the job is moved to any available server on random basis. This is introduced as primary load balancing or dispatcher-to-processor load balancing. At the point when any processor completes the job, it needs to join any I-Queue. This is designated as secondary load balancing or processor-to-dispatcher load balancing. In JIQ, two methodologies are proposed for secondary balancing: JIQ-Random and JIQ-SQ(d). In JIQ-Random, processor may join any of the dispatcher while in JIQ-SQ(d), it joins only out of the d sampled I-Queues.



. Figure. 3: Join-Idle-Queue

3. SYSTEM MODEL

Let L be the length of the queue computed over N number of dispatchers with M number of processors. Now, length of the queue for a dispatcher-queue model is given by:

$$L = \frac{M}{N} \tag{1}$$

For a system, let J be the jobs to be allocated to this dispatcher queue model such that process γ can be allocated using JIQ approach. But this method can affect the processor allocation terribly as at time t, there can be queue which is left empty that adds to the cost of the system and affect its performance. Therefore, a system S is required such that

$$S \longrightarrow f\left(N, M, J, \gamma\right) \tag{2}$$

where

$$L^s = \left\{x : x = length(Q^t) | x \neq 0\right\} \tag{3}$$

Here, Q is the queue. For ideal state, $L^s = 0$. A system closer to ideal state will be treated as a solution to the problem of secondary load balancing provided the below given constraints hold.

3.1 System Constraint

Probability of queue P being empty must follow a global minima principle and should be least at any given instance i.e. at time t, for a defined state,

$$\lim_{x \longrightarrow \max} P_i^t \longrightarrow \min \tag{4}$$

The probability P_i^t for the i^{th} queue at time t per dispatcher will be computed over the deviation of system from its normal state such that

$$P_i^t = \frac{\sqrt{\frac{1}{K}\sum_{i=1}^{K}(Q_K^t - Q_E^t)^2}}{\sqrt{\frac{1}{K}\sum_{i=1}^{K}(Q_K^t - Q_0^t)^2}} \tag{5}$$

where Q_E is the empty queue slots, Q_K is the actual queue slot, and Q_0 is the actual operating queue. For an ideal state, $Q_o^t = 0$, hence,

$$P_{i,ideal}^t = \frac{\sqrt{\frac{1}{K}\sum_{i=1}^{K}(Q_K^t - Q_E^t)^2}}{\sqrt{\frac{1}{K}\sum_{i=1}^{K}(Q_K^t)^2}} \tag{6}$$

Now, for a continuous process

$$\int_{0,t\in T}^{t} P_i^t dt \le \min((5),(6)) \tag{7}$$

Here, minimum is defined as either (5) or (6) depending upon the requirement.

3.2 Operating Cost

For the considered system, jobs are considered to be following Poisson distribution. According to this statement, operating cost i.e. time to allocate the job should be minimum or equal to job processing time only. For job y, $y \in J$, time required for completion will be computed as:

$$T_y = t_y(A) + t_y(P) \tag{8}$$

where $t_y(A)$ is the allocation time and $t_y(P)$ is the processing time. Let T_h be the threshold defined for the considered model above for which the P_i may increase due to overflow of queue. Therefore, for the defined system,

$$T_y \le T_h \tag{9}$$

Hence, processor following Poisson distribution will be defined as:

$$P_f^t = \frac{J(t)^{M_a} e^{-J(t)}}{M_a!} \tag{10}$$

where M_a denotes the actual processors available such that $M_a \in M$ and $J(t)$ is the number of jobs at time t. Now the total time will be computed as:

$$T_T = \sum_{i=1}^{k}(P_f^t) \times t + \sum_{i=1}^{k}(t(A)) \tag{11}$$

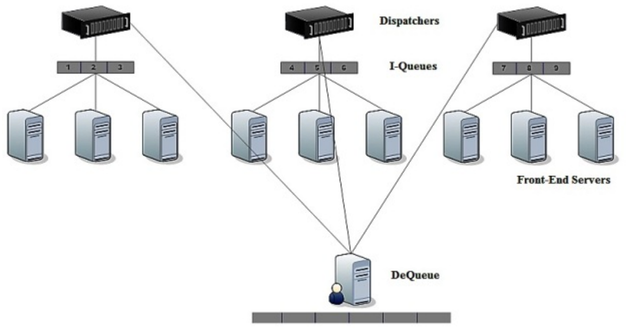
For the overall system

$$\frac{T_T}{I} \le T_h \tag{12}$$

where I is the number of iterations.

4. OUR APPROACH

We have proposed the solution for load balancing in the inverse direction from processor to dispatcher. In JIQ, a job can arrive at dispatcher whose I-Queue is vacant, while there exist idle processors in neighboring I-Queues. In Secondary load balancing situation, JIQ-Random and JIQ-SQ(d) have ample chances that a processor will not join a vacant I-Queue. Our rationale is to guarantee that each I-Queue ought to have idle processors. To accomplish this, we are utilizing following architecture which is inspired by JIQ. In JIQ, an I-Queue is maintained by



. Figure. 4: JIQ-JSQ

every dispatcher. The I-Queue is empty when all the servers are occupied. To keep up this consistency, when a server completes its job, it should join the dispatcher with a void I-Queue. A dequeue is maintained to track the dispatchers whose I-Queue is vacant. When any processor joins any dispatcher by advising the dequeue, if the I-Queue is void, the dispatcher ID is entered into the dequeue. As soon as the processor completes the current job, it refers the dequeue and joins the relevant dispatcher.

5. ANALYSIS

The main challenge in join idle queue algorithm is sending processor back to dispatcher. Sending idle servers to I-Queues consequently helps in primary load balancing. Till date, two well established algorithms Random and SQ(d) have been used. In our methodology, we have used a server to make it work like JSQ to fill the empty I-Queue.

$$JSQ. \lim_{n\to\infty} \lambda_0, R_d = \frac{\lambda}{1-\lambda}$$

$$Random. \lambda_{0,R_1} = \lambda$$

$$SQ(d). \lim_{n\to\infty} \lambda_{0,R_d} = \lambda\frac{1-\lambda^d}{1-\lambda} = \lambda + \lambda^2 + \cdots + \lambda^d$$

5.1 Backend Load Balancing

When a front end server becomes free, it sends its id to the dispatcher which then sends the processor to the I-Queue. We call the algorithm JIQ-JSQ by which there is an improvement in secondary load balancing system. At the point when a front end server turns out to be free, it sends its id to the server which then sends the idle processor to the I-Queue.
Let ρ be the proportion of occupied I-Queues in a system with n servers in equilibrium.
For JIQ-Random:

$$\frac{\rho}{1-\rho} = r(1-\lambda) \tag{13}$$

For JIQ-SQ(d):

$$\sum_{i=1}^{\infty} \rho^{\frac{d^i-1}{d-1}} = r(1-\lambda) \tag{14}$$

Similarly by putting the factor d in JIQ-JSQ, $\rho = r(1-\lambda)$ (proportion of occupied I-Queues). Arrival rate to idle server in case of JIQ-JSQ

$$= \frac{\lambda\rho}{1-\lambda} + \frac{\lambda(1-\rho)}{1-\lambda} \tag{15}$$

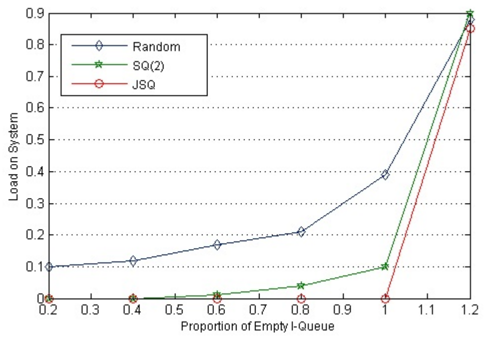
$$= \frac{\lambda}{1-\lambda}(\rho + 1 - \rho) \tag{16}$$

$$= \frac{\lambda}{1-\lambda} \tag{17}$$

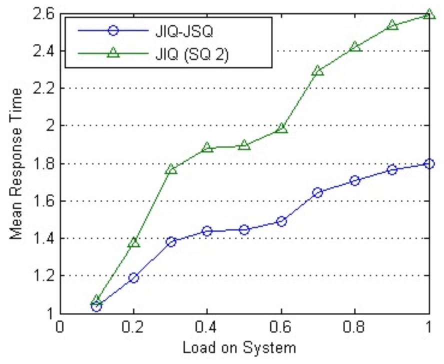
Hence, the arrival ratio of occupied server is $\frac{\lambda(1-\rho)}{1-\lambda}$ which is $\frac{1}{1-\rho}$ times greater to idle server than occupied server. We observe a substantial reduction in proportion of empty I-Queue, $n = 500$, $m = 50$, $r = 10$ and $\lambda = .6$

Table I. Proportion of empty I-Queues

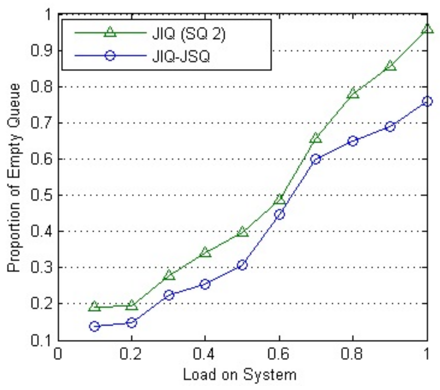
Algorithm	**Values**
Random	0.2
SQ(2)	0.027
JSQ	0



. Figure. 5: JIQ-JSQ



. Figure. 6: Mean Response Time: JIQ-JSQ and JIQ (SQ 2)



. Figure. 7: Empty I-Queue: JIQ-JSQ and JIQ (SQ 2)

In JIQ literature, SQ(2) methodology is suggested well than the random methodology. Yet, Table 1 demonstrates improvement in the extent of empty queues with our JIQ-JSQ approach as compared to other available ones. At less or sensible load, it demonstrates the proportion to be null. Till the load greater than 0.9, Figure 5 shows null with respect to the empty queues. For load till 0.9, there is no empty I-Queue. If in case it's greater than 0.9, when there are less no of idle processors than the number of I-Queue, our approach will follow the random algorithm to 1. Proportion of empty I-Queues with $r = 10$ ($m = 50$ & $n = 500$) and load is varied from 0.2 to 1. It compares the proportion of empty queues of Random, SQ(2) & JSQ. Note that the proportion of empty queues is nearly nil in case of JSQ till load 0.9. Figure 6 shows the analysis between the JIQ-JSQ(proposed approach) and the JIQ(SQ2). For mean response time, we have enhanced with respect to the load on the system using FIFO scheduling. The proposed approach provides a proactive solution to the JIQ problem thus, offers solution with 27% less response time. Figure 7 presents the analysis for proportion of empty queue after applicability of the JIQ-JSQ approach. Comparison shows that the JIQ(SQ 2) has 19% larger proportion of the queue being empty than the proposed approach. This causes untimely delays in selection of processor and job has to wait for longer duration.

6. CONCLUSION

We propose the JIQ-JSQ algorithm for the backend load balancing in distributed network. JIQ-JSQ is better than both JIQ-Random and JIQ-SQ(d). This algorithm turns out to be helpful at high load. By guaranteeing the availability of server in each I-Queue, the approach will likewise enhance the backend(dispatcher-to-processor) load balancing.

REFERENCES

Applewhite, H. L., Garg, R., Jensen, E. D., Northcutt, J. D., and Sha, L. 1982. Decentralized resource management in distributed computer systems. *Technical report, DTIC Document*.

Bramson, M., Lu, Y., and Prabhakar, B. 2010. Randomized load balancing with general service time distributions. *ACM SIGMETRICS Performance Evaluation Review 38*, 275–286.

Chou, T. C. and Abraham, J. A. 1982. Load balancing in distributed systems. *Software Engineering, IEEE Transactions on 4*, 401–412.

Gupta, V., Balter, M. H., Sigman, K., and Whitt, W. 2007. Analysis of join-the-shortest-queue routing for web server farms. *Performance Evaluation 64,* 9, 1062–1081.

Lu, Y., Xie, Q., Kliot, G., Geller, A., Larus, J. R., and Greenberg, A. 2011. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation 68,* 11, 1056–1071.

Squillante, M. S. and Nelson, R. D. 1991. Analysis of task migration in shared-memory multiprocessor scheduling. *ACM 19.*

Vvedenskaya, N. D., Dobrushin, R. L., and Karpelevich, F. I. 1996. Queueing system with selection of the shortest of two queues: An asymptotic approach. *Problemy Peredachi Informatsii 32,* 1, 20–34.

Wang, Y.-T. and Morris, R. J. 1985. Load sharing in distributed systems. *Computers, IEEE Transactions on 100,* 3, 204–217.

Kuldeep Sharma is Assistant Professor in Department of computer science & Engineering , Maharaja Agrasen University, Himachal Pradesh, India. He received his M.E degree from Department of Computer Science and engineering Thapar University, Patiala India. He is currently a Ph.D. candidate at the Department of computer science and Engineering, Thapar University Patiala, India. He is Professional member of IEEE Computer Society, IEEE Education Society and ACM Sigact.

Dr.Deepak Grag is PhD in Computer Science and Engineering with expertise of Data Science and Inteliigent Systems. He is an Chair, Computer Science and engineering Department, Thapar University Patiala, India. He is having 18 years rich cross-functional experience in continuously delivering in the capacity of teacher and researcher. Esteemed member of several professional organizations, editorial board of various journals and 108 publications to the credit. He is Chair, IEEE Computer Society India Council and Chair, IEEE Education Society India Council and Chair, ACM SIGACT North India. He is serving as Member, Board of Governors, IEEE Education Society. He is recipient of funding from National and International Agenciens to the tune of 1.30 Crores INR. He has Guided 8 PhD students and 34 Master students.