Computing and Informatics, Vol. 37, 2018, 1184–1208, doi: 10.4149/cai_2018_5_1184

A STOCHASTIC ADJUSTMENT STRATEGY FOR COORDINATION PROCESS IN DISTRIBUTED NETWORKS

Pingting Hao, Liang Hu
Jingyan Jiang, Xilong Che

College of Computer Science and Technology
Jilin University
Changchun, 130000, P. R. China
e-mail: {haopt15, jiangjy14}@mails.jlu.edu.cn,
{hul, chexilong}@jlu.edu.cn

Abstract. Cloud computing has become a popular basis that integrated into amount of large platforms to support applications (e.g., multimedia, vehicle traffic, and IoT). It is critical to focus on coordinating the part of these applications that execute in the cloud to provide reliable, scalable and available services. Nevertheless, the problem of optimally coordinating the applications is rarely addressed. In this paper, we develop a stochastic model to analyze the fundamental characteristics that occur in ZooKeeper during the coordination process. The model primarily addresses two aspects: demands of followers and the load of a leader. Then, we derive the optimal strategy for provision with deployment of coordinated servers to achieve load balancing based on various factors (e.g. server capacity and network load), so that the overall network performance is optimized. We evaluate our algorithm under realistic settings and reveal the trend of factors such as CPU, memory utilization and network bandwidth with the increasing number of requests. We propose the algorithm that considers how many servers should be deployed and when. Our results demonstrate that the strategy guarantees the performance by making suitable deployment adjustment.

Keywords: ZooKeeper, deployment, load balancing, queuing theory

Mathematics Subject Classification 2010: 49-K30, 60-G07

1 INTRODUCTION

The popularity envisioned for the fifth generation (5G) mobile network has transformed cloud services into the basic infrastructure for large-scale distribution systems. To date, not only traditional applications, but also numerous other architectures (e.g. SDN [1] and CDN [2]) have been integrated into the cloud. This rapid growth has posed significant challenges to the cloud, e.g. how to provide distributed synchronized services while maintaining configuration information to support more scalable, available and reliable services for users.

To address these challenges, ZooKeeper [3] which includes built-in, large-scale, coordinated mechanisms has been widely adopted in the cloud. The shared hierarchal namespace and data model are used for coordinating the transactions. Generally, ZooKeeper runs on several closely located servers, namely, a ZooKeeper cluster. Different roles are assigned to the servers in order to communicate with each other, to provide the functions of fault-tolerance and to deal with the information about the transactions according to the Paxos algorithm. Thus, ZooKeeper could provide a set of guarantees (e.g. sequential consistency, reliability and atomicity). ZooKeeper is typically deployed and operated by third-party architectures (e.g. HBase [4] and Storm [5]), and it is also widely used in the applications [6, 7, 8]. Such mechanisms have contributed to ZooKeeper's increasing popularity, particularly because of its distributed capabilities. Thus, achieving an optimal performance and better utilization of ZooKeeper is important also for these applications. In some cases from the companies (e.g. Baidu, Alibaba), the poor performance would not be solved only by expanding the size of cluster according to the SLA, and the results are caused by improper operations, such as the way of storage, the assignment of roles and the deployment of ZooKeeper. Thus, we pick the problem for deployment of ZooKeeper, which is one of the improper mistakes, to analyze it and find the solution.

Okorafor and Patrick [9] proposed the model based on the Hadoop/MapReduce and ZooKeeper, and utilized the ZooKeeper to coordinate the transactions instead of traditional MPI (message passing interface). Although it analyzed the number of servers for ZooKeeper, the primary goal was designed for the availability to achieve better performance (e.g. the repair rates for software and hardware, the initial failure rate, and the number of available servers). Specific to the characteristic and analysis of the ZooKeeper itself, it has no more deep research on this field. Pham et al. [13] evaluated ZooKeeper's performance under different situations (e.g. resilient name service, dynamic system configuration, and recovery databases). Complementary to [13], we combine the servers' states with ZooKeeper's performance. Furthermore, we provide an in-depth study of ZooKeeper, specific to its deployment.

In this paper, we propose the deployment for ZooKeeper according to the coordinating process. In view of the analytical results, the plan comprehensively considers the trade-off between the leader and followers with the load balancing, so that to guarantee the performance of the system. We can summarize our contributions as follows:

1. We give a brief description of the ZooKeeper workflow and develop a stochastic model to describe the coordination process using two modes: a simple mode and a general mode. Meanwhile, our stochastic model considers two conflicting roles to determine the server deployment by introducing load balancing as the metric, which provides a general baseline for understanding ZooKeeper deployment.
2. We observe the dynamic server states and their performance with dealing with requests, and derive a strategy for provisioning the optimal number of servers and determining when to adjust the deployment by considering the server states and the request distribution in the system.
3. Through a numerical analysis, we observe the performance variations: the bandwidth becomes more sensitive as the number of requests increases, and the CPU load also varies significantly. At a threshold, the throughput influence holds steady for both the leader and followers. When the service utility achieves optimality from a global viewpoint, we change the deployment to guarantee the quality of service.

The rest of this paper is organized as follows. In Section 2 we present related work. In Section 3, we describe the workflow of ZooKeeper which combines with our challenge, and provide the definition of our model. In Section 4, we concentrate on building the model for the coordination process and solving the problems mentioned in Section 3, and the results of experiments are discussed in Section 5. Finally, we conclude the paper in Section 6.

2 RELATED WORK

Recently, in the emerging coordinated services field, ZooKeeper's management consistency exhibits promising potential that can significantly improve system efficiency. Cai et al. [6] proposed a decentralized vehicle routing service system and used ZooKeeper to establish the coordination system between subtask processors. Goel and Majumdar [7] presented mutual exclusion property in ZooKeeper to guarantee payment processing. Skeirik et al. [8] focused on security services and group key management in the cloud and used ZooKeeper's logic model to bolster anti-spam and anti-virus activities. In [10], the importance of the organization and management for the service is addressed. However, the ZooKeeper is mainly used in the cloud distributed systems to support the valuable search related services.

Researchers have expended considerable effort to improve the ZooKeeper's performance. Kalantari and Schiper [11] developed a prototype to reduce the synchronization time between ZooKeeper servers. Junqueira et al. [12] designed a crash-recovery atomic broadcast algorithm to guarantee state coordination. In 5G architecture, both homogenous and heterogenous units have been integrated into the infrastructure. While the applications are likely to possess full support for developing intelligent platforms, it is challenging to utilize ZooKeeper to guarantee the services. However, performance analysis of ZooKeeper are rarely seen in the literature.

In this section, we discuss deployment, which is closely related to our work. Deployment has been a key architectural component for many years, and the literature includes many studies on deployment. The deployment objective is to identify the most appropriate number of servers, replicas and locations to achieve various optimal solutions, such as minimizing the delay [14, 15, 21, 22], bandwidth [16], energy consumed [17, 18, 19, 24], and so on. Moreover, LP relaxation techniques are widely used as a practical method to approximate the optimal solution [18]. Heuristic algorithms [16, 20] are often employed in these studies.

Our work concentrates on adjusting server deployment to guarantee the quality of service at both low cost and energy consumption. Due to the properties (e.g. intensive cluster, small sizes of items) of ZooKeeper, our approach is novel for ZooKeeper, and is more critical than the problems of location and size as the scale of components in the architecture increases. We introduce load balancing into our model. As a distributed service system, load balancing is a non-negligible factor in achieving system optimal performance [23]. Kim [24] improved a novel load balancing scheme that balances the energy consumption of the sensor nodes and the maximum network lifetime by applying sub-network management in wireless sensor networks. Bui et al. [25] presented approaches to improve networking performance by rebalancing the load on the physical links of a supercomputer. Thus, our model combines ZooKeeper with the system seamlessly, and providing insights for addressing ZooKeeper deployment.

3 SCENARIOS AND PROBLEM STATEMENT

3.1 ZooKeeper and Various Scenarios

The architecture for a generic example, the coordination process, is shown in Figure 1. Each server in the ZooKeeper cluster is configured with the service. The system involves three roles: *leader*, *follower* and *observer*. Due to the semi-distributed manner of ZooKeeper, the coordination service needs a role for the collection, computing and issuing decisions, that is the responsibility of *leader*. The number of leader is limited to one. The other two roles *follower* and *observer* mainly connect with the user, and receive the requests from one part of the area. The *follower* is responsible for satisfying the needs of users within the range, as the *observer*. The number of them is not limited.

The whole servers of *follower* and *observer* could cover the range of needs, and the number of them is not the same with *leader* that is limited to one. The part of the requests passed on to the *leader*, the other part would be dealt with by the *follower* and *observer*. The *observer* functions similarly like the followers. The difference between observers and followers is that the *observers* provide only reading services, but do not vote. Therefore, in this paper, because of their popularity and universality, we primarily discuss the first two roles.

Figure 1 shows the coordination process through the six steps. The client makes a request (step 1) to a follower. Then, the follower takes different actions depending

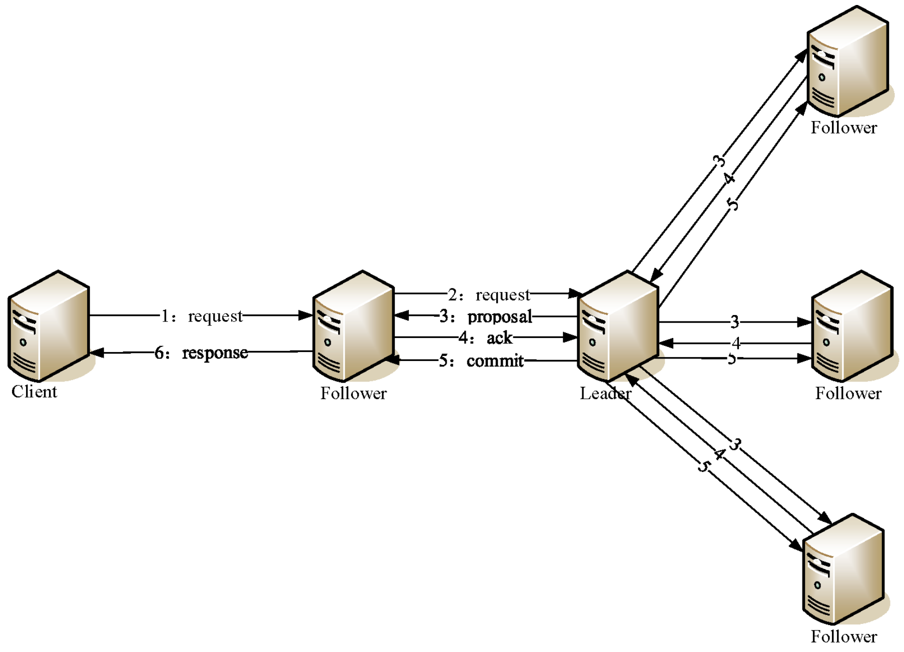


Figure 1. The coordination process (six steps) between a leader and its followers using the TCP/IP protocol

on the request type. The requests can be categorized as *read* or *write*. When the request type is *read*, the follower serves the request itself by performing step 6. When the type of the request is *write*, it is forwarded to the leader (step 2). The packet categories resulting from the write request consist of PING, REQUEST, ACK and REVALIDATE packets. The results will be returned by the followers from the leader (step 5). When the leader receives the request, it proposes a vote among its active followers (step 3). After the vote is returned by each follower (step 4), the leader calculates the result and notifies all the followers (step 5). The packet types implemented by each follower including PROPOSAL, COMMIT, UPDATE and REVALIDATE, and the request procedure obeys the coordination process. The protocol Zab, derived from the Paxos algorithms, is the basis of the coordination process that makes the system reliable, consistent and available. For example, if the leader crashes, the remaining followers elect a new leader based on the Zab protocol. In addition, when the leader performs update operations, it propagates the incremental state changes to its followers so that the consistency is guaranteed. Note that there is an odd number of servers ($2n+1$, $n = 1, 2, 3, \ldots$), and the number of active servers is never less than $n + 1$, which avoids evenly splitting votes.

3.2 Problem Definition and Notation

In this section, we describe the coordination model and formulate the optimization problem in ZooKeeper. Table 1 summarizes the notations used in this paper.

First, the client connects to one server to create a client/server session. Then, the client can submit requests to the server. Next, the server receives the client's requests and judges the type of requests. Based on the request type, it determines whether the request must be transferred to the leader. *Search* and *read* operations are not forwarded, while *setData*, *create* and *delete* requests among others are forwarded to the leader. For the latter, the leader organizes voting among its active followers and returns the results to all followers as a *commit* operation. Clients receive their results from the followers. Thus, the challenge is rising from this procedure. The amount of requests are received by the follower, and the more requests need the followers with more capacity to deal with such as increasing the number of followers, so that the burden of followers could be afforded and guarantee the speed of dealing with the requests. However, the effect of the leader is opposite from the followers due to the semi-distributed manner of ZooKeeper. The more requests and actions would result in the more burden of the leader, and the performance would be influenced by the leader. If the leader is also responsible for the work of the followers, it would be worse for the performance. Generally, the leader is not set to receive the users' requests as default. Thus, the scaling point of the cluster could be considered from the two angles that is dealing with the requests by the followers and the leader.

The network topology is represented as a graph $G = (V, E)$. V contains n nodes connected by the edges from set E. Based on the definition of a Poisson process, we assume that the arrival rate of clients λ follows a Poisson distribution, which has been discussed in many papers. For the departure rate of the clients μ, the instances can be separated into two different situations. The first case is simple, and client requests are served at rate ν under ideal conditions. In the second case, we consider a realistic scenario in which failure is introduced. We denote the failure rate of the leader as f_{sL} and the failures caused by clients is denoted as γ, where γ is a part of μ. At the physical level, many incidents can cause the system to become unavailable, such as power outages, wire disconnections, routing inaccuracies, link errors, and so on. Except for special and infrequent situations, we focus on the link error rate f_l.

Next, we set the network parameters. M defines the total number of servers, including the leader and the followers, while m denotes the number of followers. Considering the limited real-world conditions, we assume that client bandwidth is all the same, and the upper bound of the bandwidth is c_1. The followers have a maximum bandwidth of c_2, while the upper bound of the leader bandwidth is c_3. In this paper, we also add the servers' attributes to this model. Combined with the theory of load balancing, FW stands for the load capacity of the follower, and LW denotes the leader's load capacity. Load capacity is reflected by the CPU idle time V_1, the surplus storage space V_2, and the remaining network bandwidth V_3. However, the two types of requests, *read* and *write*, have different effects on server load capacity. Thus, we denote that the number of requests for two types as rw_1 and rw_2 to show their respective influences on ZooKeeper.

NOTATION	DEFINITION
λ_{si}	Client arrival rate to server i, $i = 1, 2, 3, \ldots$, represents the followers, and $i = L$ is the leader
ν	Client departure rate under ideal conditions
μ	Actual client departure rate
ρ	The intensity of service, $\rho = \lambda/\mu$
γ	The portion of the departure rate due to a client fault
f_{sL}	The leader sL failure rate
f_l	The link l failure rate
W	The average client waiting time (response time)
N_{si}	The number of operations that pass on server i, where $i = L$ represents the leader
rw_i	The number of requests, $i = 1$ denotes the type of requests *read*, $i = 2$ denotes the type of requests *write*
P_n	$P_n = P\{N = n\}$ $(n = 0, 1, 2, \ldots)$ denotes the event probability when the number of clients n is in a stable state.
$D(x)$	Throughput is counted as the number of packets that pass through a server in the interval x
S_x	The rate of transmission in the interval x
LW_{sL}	The performance of server L, referring to the leader using factors v_i to describe them
FW_{si}	The performance of server i, referring to the follower using factors v_i to describe them
c_i	The bandwidth constraint. The assigned bandwidth of the clients is expressed as $i = 1$, $i = 2$ denotes the maximum follower bandwidth, and $i = 3$ denotes the maximum leader bandwidth
v_i^{si}	The measurement of server i's performance concerning factor i

Table 1. The major symbols used in this paper

4 COORDINATION MODEL

In this section, we develop the strategy to adjust ZooKeeper deployment and provide two models for analyzing the coordination process.

4.1 Simple Model

We first consider a simple model for ZooKeeper without considering the unexpected accidents that occur in the real conditions. Based on the overall situation, the coordination model is assumed to be M/M/1. In addition, FCFS (first-come-first-serve) is the queuing discipline model in ZooKeeper.

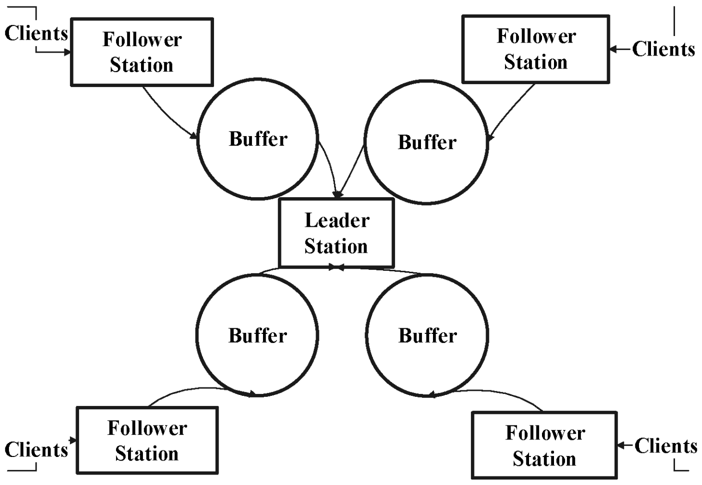


Figure 2. The simple architecture used for the queuing theory

As shown in Figure 2, the leader is the central pivot in the system and is also the organizer when refereeing a proposal. Thus, the buffer of each follower is filled with requests that the leader must address. According to queuing theory, the arrival rate for the followers i is λ_{si} $(i = 1, 2, 3, \ldots, m)$. Requests from follower i follow the probability σ_{si} $(i = 1, 2, 3, \ldots, m)$ to the leader, and their arrival rate at the leader λ_{sL} is defined in Equation (1):

$$\lambda_{sL} = \frac{rw_2}{rw_1 + rw_2} \times \sum_{i=1}^{m} \sigma_{si}\lambda_{si}. \tag{1}$$

According to Little's law [26], we consider the average waiting time for one follower in Equation (2):

$$W = \frac{1}{\nu - \lambda}. \tag{2}$$

We assume that a client submits a request to the server and does not submit another request until the result is returned to the client. Therefore, the number of operations N_{sL}, as shown in Equation (3), means that the operation has passed through the leader. The number of operations assigned to the followers is denoted by N_{si} in Equation (4). We also assume that each request occupies identical bandwidth and that bandwidth capacity is limited, which means that the total bandwidth used between the clients and followers during a given interval should be less than the followers could provide in practice. We apply the constraint shown in Equation (5):

$$N_{sL} = \lambda_{sL} \times W, \tag{3}$$

$$N_{si} = \lambda_{si} \times W, \tag{4}$$

$$c_1 \times N_{si} \leq c_2. \tag{5}$$

4.2 General Model

In practice, failure cases are considered in the general model. This model supplements the simple model and is closer to reality. There are three types of failure: physical failure, leader-caused failure and client-caused failure. The physical failure rate f_l denotes the link error rate. The leader's failure rate f_{sL} results in reelection. When the leader is out of control, the followers first terminate their connections with the clients. Then, the followers vote for the role of a new *leader*. Each follower has at least one connection with the others. The votes are marked with *zxid* to represent the transaction state, and the votes also have the *server id* that reflects the source. When a server gets a majority of votes (more than half), that server is chosen to be the new leader and reestablish the coordination process in ZooKeeper. The system may experience less throughput during the voting process. In addition, we account for the client's failure rate γ, which reduces the departure rate μ. The follower may cause the failure, but it will not influence the performance. Information about a client's session is stored by both the followers and the leader. When a follower failure occurs, the connected client receives a message from the follower or the TTL times out. Then, the client chooses a new active follower from the set of followers. Recovery from this type of failure is both simple and fast. Therefore, we ignore it in the remainder of this paper. Due to the coordination process, failure will be detected whatever it occurs among the *leader*, its *followers*, and their *clients*. Assuming that the starting time is $x = 0$, we define the number of operations assigned to followers in Equation (6):

$$N_{si} = \lambda \times W - \int_{x=0}^{x=t} \gamma \,\mathrm{d}x - \int_{x=0}^{x=t} f_l \,\mathrm{d}x - f_{sL} \times \lambda_{si} \times W. \tag{6}$$

The two types of requests, *read* and *write*, reflect the different coordination process shown in Figure 1. Therefore, we must compute the number of packets for the two request types differently. The number of packets $D(x)$ $(x = t_1, t_2, t_3, \dots, t_n)$ in an interval is defined in Equation (7). Using the number of packets $D(x)$, the rate of transmission S_x can be measured indirectly, as shown in Equation (8):

$$D(x) = \frac{rw_2}{rw_1 + rw_2} \times N_{si} \times [6 + 3 \times (M-2)] + \frac{rw_1}{rw_1 + rw_2} \times N_{si} \times 2, \tag{7}$$

$$S_x = \frac{dD(x)}{dt}. \tag{8}$$

Then, we introduce the Markov birth-death process to describe the dynamic client changes. Assumed that $\lambda_{n-1} = \lambda_{n-2} = \ldots = \lambda_0$, and $\mu_n = \mu_{n-1} = \ldots = \mu_1$. Then, the probability of an empty queue can be derived under the constraint of $\sum P = 1$ as shown in Equation (9):

$$P_0 = \frac{1}{1 + \sum_{n=1}^{\infty} \rho^n}. \tag{9}$$

The variable *ns* denotes the probability of success for completing the coordination process. Moreover, the bandwidth limit c_{max} should not be negligible. Furthermore, if the type of request is *read*, then the M/M/1 queuing system transforms into M/M/1/c_{max} during the read. However, the *write* type rarely reaches the upper bound of the bandwidth because of its order in the leader. Thus, we assume that *write* type requests still use the M/M/1 queuing system. Then, we have the following:

$$\begin{aligned} ns = {} & \frac{rw_2}{rw_1 + rw_2} \times (1 - f_l) \times (1 - f_{sL}) \times (1 - P_0^{rw_1}) \\ & + \frac{rw_1}{rw_1 + rw_2} \times (1 - f_l) \times (1 - f_{sL}) \times (1 - P_0^{rw_2}). \end{aligned} \tag{10}$$

In order to simplify the probability of an empty queue, we transform the P_0 as (10) into (11) using the queuing theory of M/M/1.

$$P_0^{rw_1} = \sum_{n=1}^{\infty} P_n = 1 - P_0 = 1 - \rho. \tag{11}$$

Then, we separate the formula (10) into two parts according to the different types of requests. By using the simplified formulation (11), we conclude the probability of success for completing the type of request *write* in Equation (12):

$$ns_1 = \frac{rw_2}{rw_1 + rw_2} \times (1 - f_l) \times (1 - f_{sL}) \times \rho. \tag{12}$$

The probability of an empty queue for the type of request read obey with the theory of M/M/1/c_{max}, and the probability in (10) is simplified into (13). Thus, the probability of success for completing the type of request *read* is computed as

$$P_0^{rw_2} = \rho^i \times \frac{1 - \rho}{1 - \rho^{c_{max}+1}} (i = 0, 1, 2, \ldots, c_{max}), \tag{13}$$

$$ns_2 = \frac{rw_1}{rw_1 + rw_2} \times (1 - f_l) \times (1 - f_{sL}) \times \frac{1 - \rho}{1 - \rho^{c_{max}+1}}. \tag{14}$$

Next, we consider how to achieve the load balancing in the system. More specifically, the follower is mainly responsible for dealing with the requests from the clients. Due to the increasing number of requests, we assume the leader is mainly responsible for the decision part without receiving the requests directly from the client. Thus, we define the capacity of followers and leader, respectively. We choose the metrics to give a comprehensive understanding of the capacity of the followers in (15), which including the CPU idleness V_1, the surplus storage space V_2, and the surplus bandwidth of access for the server V_3. Then, we set the weights k_1, k_2, and k_3 for the three metrics. Specific to ZooKeeper, the capacity for dealing with the requests would mainly influence the CPU, the storage for requests would mainly influence the memory metric, and the bandwidth would be used for communication and delivery. Thus, we take the three important metrics and set the weight of them almost the same. The constraint of the bandwidth is shown in Equation (16), which defines the minimal surplus bandwidth.

$$FW_{si} = k_1 \times V_1^{si} + k_2 \times V_2^{si} + k_3 \times V_3^{si}$$
$$(k_1 + k_2 + k_3 = 1, V_1^{si} \in (0,1), V_2^{si} \in (0,1), V_3^{si} \in (0,1)), \quad (15)$$

$$\min(V_3^{si}) = \frac{c_2 - N_{si} \times c_1}{c_2}. \quad (16)$$

Load balancing is the global theory for evaluating the system. For the assignment of workload, the regulation is to distribute jobs to the servers according to the capability. If the server has a higher capability, it could do more work than the servers which have lower capability. This arrangement can achieve the balance for the system, and such that it could improve the utilization of the resource. Thus, we give the formula (17) as follows:

$$\frac{N_{si}}{FW_{si}} \approx \frac{N_{sj}}{FW_{sj}}. \quad (17)$$

The system maintains the state shown in Equation (17) to achieve the best use of the resources for each server. We observe the indicator b_1 and show the relative load of follower i in (18). When b_1 exceeds its upper limit, $bound1$, the current number of servers in the ZooKeeper cluster is insufficient to service the requests. From a follower viewpoint, the cluster needs to scale as the number of requests increases. Otherwise, the performance will degrade:

$$b_1 = \frac{N_s}{N_{si} \times FW_{si}} \geq bound1. \quad (18)$$

The definition for the leader capacity is similar to that of the followers. We apply the constraints shown below in Equations (19) and (20):

$$LW_{sL} = k_4 \times V_1^{sL} + k_5 \times V_2^{sL} + k_6 \times V_3^{sL}$$
$$(k_4 + k_5 + k_6 = 1, V_1^{sL} \in (0,1), V_2^{sL} \in (0,1), V_3^{sL} \in (0,1)), \quad (19)$$

$$\min(V_3^{sL}) = \frac{c_3 - m \times c_2}{c_3}. \quad (20)$$

By checking the relative load of the leader b_2, as shown in Equation (21), we judge the timing for adjusting the deployment with b_1. For example, if we add some quantity of followers, the leader load will increase as the number of requests increases. A heavy leader load results indirectly in slower responses. When the load exceeds the leader's upper limit $bound2$, the number of followers in the system should be correspondingly reduced.

$$b_2 = \frac{D(x)}{LW_{sL} \times \overline{b_1}} \geq bound2 \tag{21}$$

Through the formulations in Equations (18) and (21), whether to scale the system and when to adjust the followers are the problems that we must address. We synthesize two aspects to consider this problem. After trading off the two aspects, we obtain the formulation in Equation (22):

$$\begin{aligned} b_3 &= k_7 \times b_1 + k_8 \times b_2 \\ &\quad (k_7 + k_8 = 1, b_1 \in (0,1), b_2 \in (0,1)) \end{aligned} \tag{22}$$

where the threshold is represented by b_3. The case in which $bound1$ and $bound2$ are both achieved is out of our scope because that condition denotes that the leader capacity needs to be improved to guarantee the performance. Instead, we focus on the utility when sufficient resources are available. When b_3 is achieved, the number of followers should be adjusted, either maintained, increased or decreased. The adjustment of followers could be simply described as follows:

$$b_3 = \begin{cases} \text{increase the number of followers,} & \text{beyond } bound1 \ \&\& \text{ equals } b_3 \\ \text{keep status,} & \\ \text{decrease the number of followers,} & \text{beyond } bound2 \ \&\& \text{ equals } b_3 \end{cases} \tag{23}$$

5 NUMERICAL RESULTS

In this section, we use a realistic platform to demonstrate follower reassignment as the basis for load balancing. We observe the three metrics of load balancing and the throughput in two related deployments. Then, we compute the relative loads for the followers and the leader to explore the threshold b_3. Complementary to the theoretical analysis, the experimental results both allow us to gain a better understanding of the reassignment operation and provide guidance to determine approaches which will improve the system's performance and utility.

5.1 Experimental Setup

Under the ZooKeeper platform, we evaluate our model using a random local area network topology, which means that we pick the servers randomly from the set of local network. Every server runs on Intel Core i3 and has 2 GB of RAM. We assume that the number of servers is s and configure three different deployments ('$s = 3$', '$s = 5$', and '$s = 7$') for the experiment. Then, we could make the adjustment from two groups, one group is between '$s = 3$' and '$s = 5$', the other is between '$s = 5$' and '$s = 7$'. We execute each experiment 50 times to obtain average values, and the experiments mainly involve the actions both the leader and followers take part in. The arrival rate increases by a factor of 10 until it reaches the upper limit of the servers' capacity. Although the load of each server reflects the latency, we still set an acceptable latency range of 100 ms. Additionally, in this paper, we define the latency that starts when the clients submit a request and ends when the clients obtain the results from the followers. The ramp-up period is set at 1s, during which each factor is monitored until it reaches a stable state. We intentionally set the weights k_1:k_2:k_3 = 4:3:3 for evaluating the followers' loads, and to represent the similar importance for the three metrics, as well as that of the leader. The upper limit of the bandwidth c_2 for the followers is fixed, and the upper limit for the leader c_3 is also fixed. The metrics for the failure part (e.g. the client's failure rate γ, the link error f_l , the leader's failure rate f_{sL}) are set as the realistic pattern rather than the assumed value. Thus, the throughput S_1 and the number of packets $D(x)$ could be derived. Meanwhile, the related information of CPU, memory and bandwidth would be required through monitoring with the input metrics, and the adjustment is performed after computing the bound.

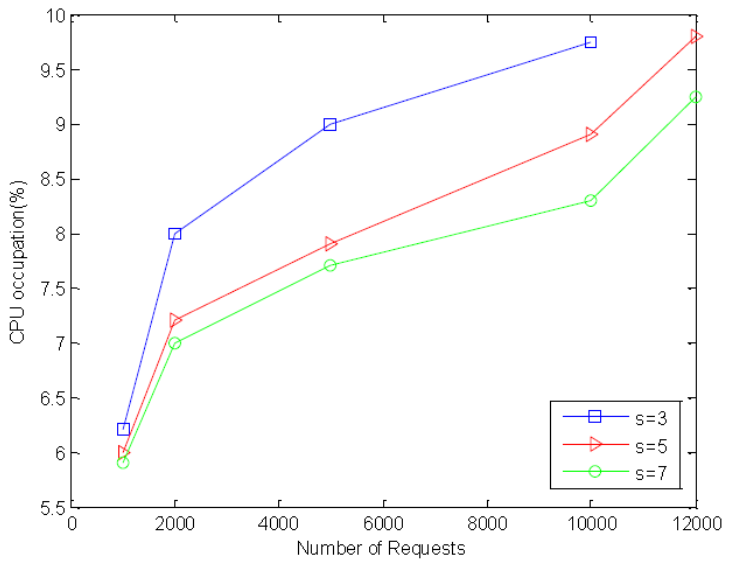
5.2 Evaluation Results

5.2.1 The Analysis for Followers

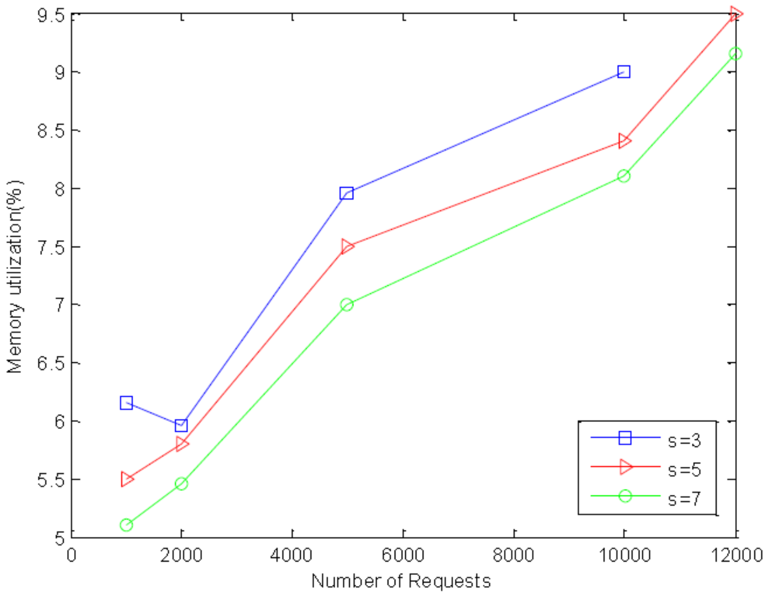
We first demonstrate the trend of three metrics to evaluate the server loads as the number of requests from each follower increases: CPU utilization (%), memory utilization (%), and network bandwidth (%). In addition, the throughput measures the number of requests in a millisecond.

The definition of x axis means that the number of requests would be submitted to the system. The results in y axis are collected from each follower and computed averagely. The results, presented in Figure 3 a), show that the trend of follower's CPU utilization as the number of requests increases. A similar trend occurs not only in the deployment as fewer servers but also more servers. The CPU utilization trend is related to the follower workloads. Thus, the CPU utilization increases with the heavier load of followers accordingly. However, the two deployments are different that more followers cause the CPU utilization of each follower to decrease with the same number of requests. When the number of requests reaches a certain point, the followers nearly achieve maximal capacity. The utility of the CPU in the system approximately achieves its extreme. This is reason for scaling the system to serve more requests.

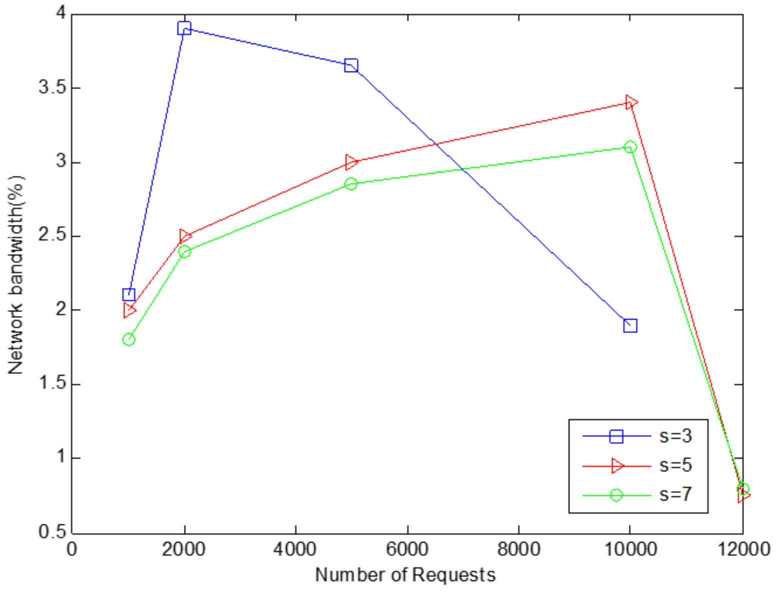
In Figure 3 b), the available memory is sufficient for the experiment. The trend of memory utilization also changes with the number of requests. The more followers there are, the total required storage space could be divided more. Moreover, the



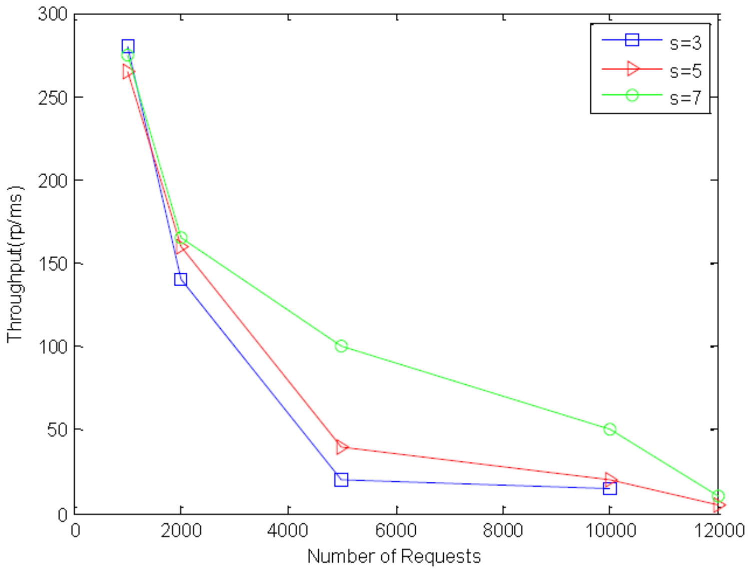
a) The trend of CPU_utilization for the followers



b) The trend of Memory_utilization for the followers



c) The trend of Network_bandwidth for the followers



d) The trend of Throughput for the followers

Figure 3. The capacity of the follower with the increasing number of requests

trend when '$s = 3$' changes more obviously due to the more requests of each follower. However, the dynamic states of network bandwidth are not only related to the number of requests, but also influenced by the network condition (e.g. congestion). The finite bandwidth capacity is one of the main reasons that caused the congestion. Another is the queuing length of each follower. Thus, as shown in Figure 3 c), when the number of requests is within a certain range, the utilization of network bandwidth expands as the number of requests increases, and the same trend occurs in network I/O. However, when the number of requests exceeds the bandwidth capacity, a decreasing trend appears due to the capacity limit. Moreover, as this trend approaches zero, the system breaks down. The peak point for the deployment '$s = 3$' comes up earlier than the other two deployment due to the more number of requests from followers, and it also means that the limit also appears earlier. The deployment '$s = 5$' and '$s = 7$' has the similar tendency in this figure. However, the average number of requests for '$s = 7$' is less than the deployment '$s = 5$', and the limit point is almost the same mainly due to the conflicting requests.

With the increasing number of requests, the trend of the throughput is shown in Figure 3 d). The figure shows that the throughput decreases when the number of requests is 1 000, 2 000 and 5 000. After that, the trend gradually stabilizes. This result is indirectly related to the resource utilization. When the received load is well below the capacity, the throughput is high. As the load of followers increases, the throughput decreases sharply. Firstly, the deployment '$s = 3$' has not been achieved the turning point, the throughput keeps higher because of the largest number of requests. Then, the throughput would decrease with some reasons (e.g. the capacity of servers, congestion, and the conflicting requests). Thus, the deployment '$s = 7$' has the highest throughput after the first point of the deployment '$s = 3$' until the limit point. When the number of requests achieves the limit point, the deployment '$s = 5$' and '$s = 7$' would converge to similar throughput.

In summary, the first time each experiment executes, there are small deviations due to the adaptive phase. The four graphs with dotted points are superimposed to simply analyze the capacities of followers and the system's performance under the three deployments. We can make the following conclusions from the findings above. First, CPU utilization is steady in each experiment. Next, memory utilization fluctuates within 20 % of full utilization. Bandwidth utilization is lower than memory utilization, which means that storage is more likely to be a bottleneck than the bandwidth in ZooKeeper.

In the '$s = 3$' deployment, the system can serve approximately ten thousand requests, but it appears to reach a non-working state when the number of requests exceeds ten thousand. This result indicates that a bound exists in ZooKeeper when served requests beyond the extreme point, and the adjustment of deployment should be considered. The deployment of '$s = 5$' has more servers to deal with the requests. Therefore, the limit expands to 12 000 requests the same as the deployment '$s = 7$'. Although the deployment '$s = 5$' and '$s = 7$' has the similar extreme point, the throughput for the deployment '$s = 7$' has the advantage than the deployment '$s = 5$', as shown in Figure 3 d). When the capacity of the follower in the system is exceeded, an adjustment must be considered to guarantee the performance.

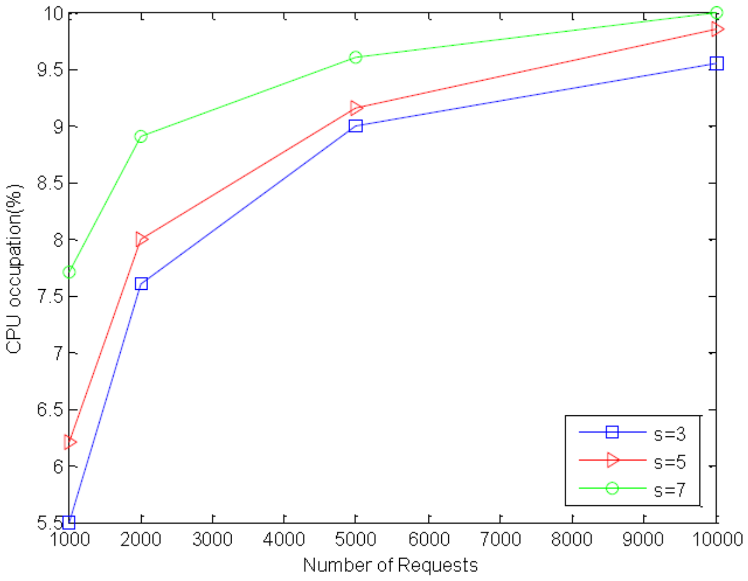
5.2.2 The Analysis for the Leader

The results presented in Figure 4 show the relative load of the leader using the three metrics. The x axis represents for the number of requests that the leader received. With regard to CPU utilization, the trend for the leader is almost similar. The deviations between the three deployments in Figure 4 a) are caused by the overhead, which produces through the different numbers of requests. With the number of followers increasing, the utilization of CPU is higher gradually. Thus, the deployment '$s = 7$' has the highest utilization than the other two deployment. For the memory utilization in Figure 4 b), the '$s = 7$' deployment needs more space than the '$s = 5$' deployment, and the '$s = 5$' deployment needs more space than the '$s = 3$' deployment. The situation is directly opposite to the trend of followers. From the leader's point of view, the reason for this disparity is that the number of followers results in the different total number of requests. Thus, this situation means that the leader must maintain more information to manage and control the system. With the requests swarming in, congestion becomes severe and leads to a lower bandwidth utilization as shown in Figure 4 c). Thus, the deployment '$s = 7$' has the lower utilization. When the number of requests from each follower achieves 5 000, the two deployments '$s = 5$' and '$s = 7$' would reach the limit, and the derivation between the two deployments is around 0.5 % utilization. When the number of requests from each follower achieves 10 000, the bandwidth for deployment '$s = 5$' and '$s = 7$' have slight increasing due to the higher number of requests. The deployment '$s = 5$' has risen around 0.04 %, and the deployment '$s = 7$' has risen around 0.02 %. Thus, the swarming does not influence much to the limit situation.

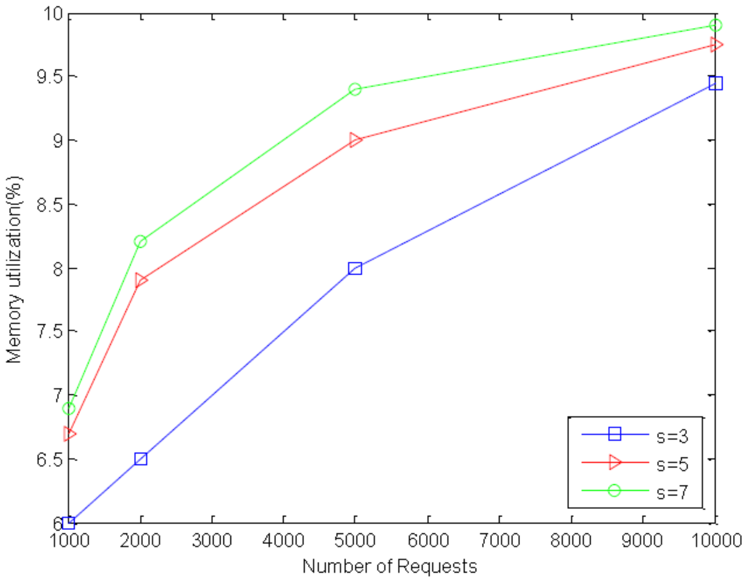
In Figure 4 d), firstly, the deployment '$s = 5$' has higher throughput than the deployment '$s = 3$' due to the higher number of requests. Then, the throughput would decrease because of the increasing load of leader, and the deployment '$s = 7$' decreases sharply from the number of requests 1 000 to the number of requests 2 000. Until the limit point which the number of requests from each follower is 5 000, the deployment '$s = 5$' and '$s = 7$' has similar throughput. Thus, when we are ready to increase the number of followers, we also need to consider the leader's state. Moreover, when the leader approaches its breakdown point, it also influences overall system performance. Thus, the adjustment timing should consider the two conflicting roles as follows.

5.2.3 The Adjustment for the System

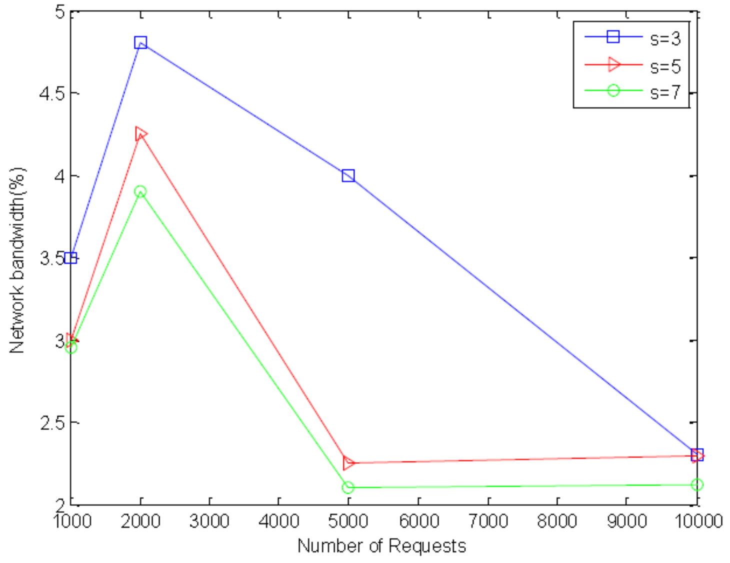
We first consider maximizing the utility in this paper. Load balancing is intended to take full advantage of the resource rather than to only consider the limits of hardware capacity. For dealing with more requests, the system needs to expand the cluster of followers. However, the more burden on the leader would cause the



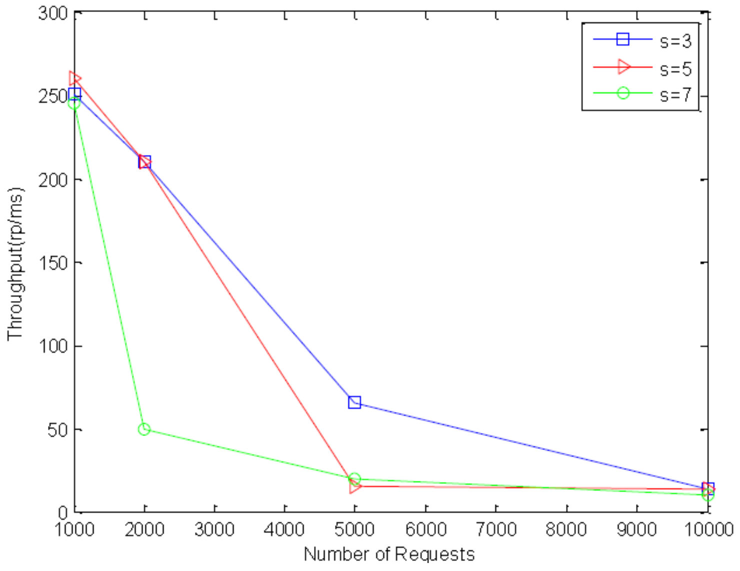
a) The trend of CPU_utilization for the leader



b) The trend of Memory_utilization for the leader



c) The trend of Network_bandwidth for the leader



d) The trend of Throughput for the leader

Figure 4. The capacity of the leader with the increasing number of requests

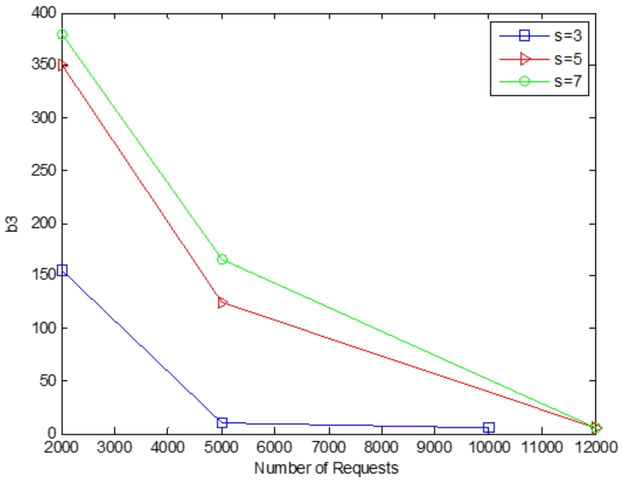


Figure 5. The tendency of the bound as the number of requests increases in ZooKeeper

worse performance. Thus, we combine the two conflicting situations to adjust the deployment. The timing for adjust could depend on various needs of applications, such as the requirement of throughput, bandwidth and memory.

Figure 5 depicts the trend of $bound3$, which is derived from $bound1$ and $bound2$. We set the proportions of k_7 and k_8 according to the proportions computed from the communication steps between the follower and the leader. During the complete

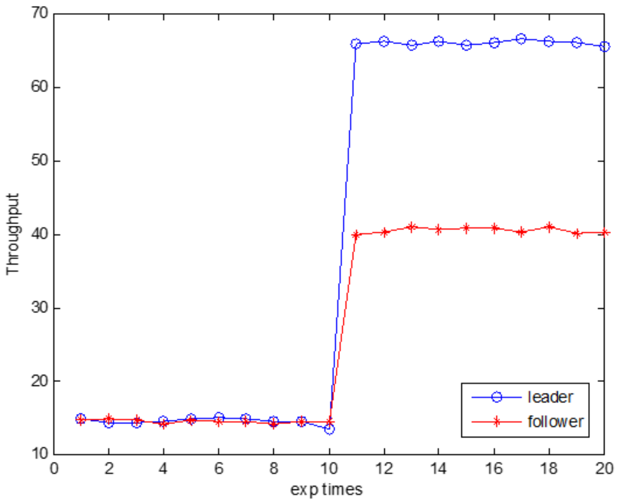


Figure 6. The throughput of the leader and followers before and after adjustment

experimental procedure, $bound1$ of the followers decreases as the system scales because of the increasing load. Then, the $bound2$ of the leader also decreases because of the realistic platform. Under the same number of requests, $bound1$ is smaller when there are fewer followers. When the number of requests is below 5 000, the trend of the '$s = 5$' and '$s = 7$' deployment decreases more sharply than that of the '$s = 3$' deployment. As the deployment of '$s = 3$', before the 5 000-request point, the number of followers could also be decreased. The trend remains stable, decreasing gradually between 5 000 requests and the extreme. This result denotes the timing that the resource utility achieves optimality. Thus, we assume that our actions should be taken after the extreme. Namely, the extreme is the point at which to adjust the followers to maintain a specific throughput volume. For example, the strategy could adjust from the deployment '$s = 3$' to '$s = 5$' as shown here. The similar adjustment could be from the deployment '$s = 5$' to '$s = 7$'. Note that the specific adjustment strategy should be further studied according to different objectives. By eliciting the leader's bound, $bound2$, and the followers' bound, $bound1$, we deduce the threshold $bound3 \approx 5.30$ for ZooKeeper in our cases. Figure 6 compares the throughput before and after the adjustment. The result demonstrates that the leader and followers are still in the steady phase and that the performance has been improved through the adjustment.

6 CONCLUSIONS

The research here provides a novel perspective for the analysis of ZooKeeper performance. In the emerging large platform, coordination services offer more capacity for the new challenge. In this paper, we developed a stochastic model to describe the coordination process. Based on this model, we derived an optimal strategy for adjusting server deployments in ZooKeeper and observed the results using a real environment. The experimental results demonstrated the trend of various server metrics, and the results revealed the sensitivity of each metric with providing insights for control schemes. Meanwhile, the influences of the performance we quantified could aid in making deployment decisions. Considering the conflicting roles involved in load balancing, thresholds are given to maximize the resource utilization. In the future, we can potentially improve system scalability, reliability and availability based on specific multimedia applications using the distributed platform.

Acknowledgement

This work is funded by: National Key R & D Plan of China under Grant No. 2017YFA0604500, National Sci-Tech Support Plan of China under Grant No. 2014BAH02F00, by National Natural Science Foundation of China under Grant No. 61701190, by Youth Science Foundation of Jilin Province of China under Grant No. 20160520011JH and 20180520021JH, by Youth Sci-Tech Innovation Leader and Team Project of Jilin Province of China under Grant No. 20170519017JH, by Key Technology Innovation Cooperation Project of Government and University for the whole Industry

Demonstration under Grant No. SXGJSF2017-4, and by Key Scientific and Technological R & D Plan of Jilin Province of China under Grant No. 20180201103GX.

REFERENCES

[1] Kreutz, D.—Ramos, F. M. V.—Veríssimo, P. E.—Rothenberg, C. E.—Azodolmolky, S.—Uhlig, S.: Software-Defined Networking: A Comprehensive Survey. Proceedings of the IEEE, Vol. 103, 2015, No. 1, pp. 14–76, doi: 10.1109/JPROC.2014.2371999.

[2] Sahoo, J.—Salahuddin, M. A.—Glitho, R.—Elbiaze, H.—Ajib, W.: A Survey on Replica Server Placement Algorithms for Content Delivery Networks. IEEE Communications Surveys and Tutorials, Vol. 19, 2017, No. 2, pp. 1002–1026, doi: 10.1109/COMST.2016.2626384.

[3] Hunt, P.—Konar, M.—Junqueira, F. P.—Reed, B.: ZooKeeper: Wait-Free Coordination for Internet-Scale Systems. USENIX Annual Technical Conference, Vol. 8, 2010, p. 9.

[4] Apache HBase. Availaible at: `http://hbase.apache.org/`.

[5] Marz, N.: Storm: Distributed and Fault-Tolerant Realtime Computation. Available at: `https://www.storm-project.net/`, 2015.

[6] Cai, M.—Liang, C.—Chen, W.—Su, H.: Realtime Vehicle Routes Optimization by Cloud Computing in the Principle of TCP/IP. 10th International Conference on Service Systems and Service Management (ICSSSM '13), IEEE, 2013, pp. 113–118, doi: 10.1109/ICSSSM.2013.6602650.

[7] Goel, L. B.—Majumdar, R.: Handling Mutual Exclusion in a Distributed Application Through ZooKeeper. International Conference on Advances in Computer Engineering and Applications (ICACEA '15), IEEE, 2015, pp. 457–460, doi: 10.1109/ICACEA.2015.7164748.

[8] Skeirik, S.—Bobba, R. B.—Meseguer, J.: Formal Analysis of Fault-Tolerant Group Key Management Using ZooKeeper. 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2013, pp. 636–641, doi: 10.1109/CCGrid.2013.98.

[9] Okorafor, E.—Patrick, M. K.: Availability of JobTracker Machine in Hadoop/MapReduce ZooKeeper Coordinated Clusters. Advanced Computing, Vol. 3, 2012, No. 3.

[10] Jiang, C.—Ding, Z.—Wang, P. et al.: An Indexing Network Model for Information Services and Its Applications. 2013 IEEE 6th International Conference on Service-Oriented Computing and Applications, 2013, pp. 290–297.

[11] Kalantari, B.—Schiper, A.: Addressing the ZooKeeper Synchronization Inefficiency. In: Frey, D., Raynal, M., Sarkar, S., Shyamasundar, R. K., Sinha, P. (Eds.): Distributed Computing and Networking (ICDCN 2013). Springer Berlin Heidelberg, Lecture Notes in Computer Science, Vol. 7730, 2013, pp. 434–438.

[12] Junqueira, F. P.—Reed, B. C.—Serafini, M.: Zab: High-Performance Broadcast for Primary-Backup Systems. 2011 IEEE/IFIP 41st International Confer-

ence on Dependable Systems and Networks (DSN '11), 2011, pp. 245–256, doi: 10.1109/DSN.2011.5958223.

[13] Pham, C. M.—Dogaru, V.—Wagle, R. et al.: An Evaluation of ZooKeeper for High Availability in System S. Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE '14), 2014, pp. 209–217.

[14] Abdel-Rahman, M. J.—Mazied, E. D. A.—Teague, K. et al.: Robust Controller Placement and Assignment in Software-Defined Cellular Networks. 26th International Conference on Computer Communication and Networks (ICCCN '17), IEEE, 2017, pp. 1–9.

[15] Marotta, A.—D'Andreagiovanni, F.—Kassler, A.—Zola, E.: On the Energy Cost of Robustness for Green Virtual Network Function Placement in 5G Virtualized Infrastructures. Computer Networks, Vol. 125, 2017, pp. 64–75.

[16] Chen, J.-B.—Chen, C.-C.: Using Particle Swarm Optimization Algorithm in Multimedia CDN Content Placement. Fifth International Symposium on Parallel Architectures, Algorithms and Programming (PAAP '12), IEEE, 2012, pp. 45–51, doi: 10.1109/PAAP.2012.15.

[17] Jayasundara, C.—Nirmalathas, A.—Wong, E.—Chan, C.: Improving Energy Efficiency of Video on Demand Services. IEEE/OSA Journal of Optical Communications and Networking, Vol. 3, 2011, No. 11, pp. 870–880, doi: 10.1364/JOCN.3.000870.

[18] Llorca, J.—Tulino, A. M.—Guan, K. et al.: Dynamic In-Network Caching for Energy Efficient Content Delivery. INFOCOM, 2013, pp. 245–249, doi: 10.1109/INFCOM.2013.6566772.

[19] Choi, N.—Guan, K.—Kilper, D. C.—Atkinson, G.: In-Network Caching Effect on Optimal Energy Consumption in Content-Centric Networking. 2012 IEEE International Conference on Communications (ICC '12), 2012, pp. 2889–2894, doi: 10.1109/ICC.2012.6364320.

[20] Lange, S.—Gebert, S.—Zinner, T. et al.: Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks. IEEE Transactions on Network and Service Management, Vol. 12, 2015, No. 1, pp. 4–17.

[21] Ramesh, S.—Rhee, I.—Guo, K.: Multicast with Cache (MCache): An Adaptive Zero-Delay Video-on-Demand Service. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 11, 2001, No. 3, pp. 440–456, doi: 10.1109/INFCOM.2001.916690.

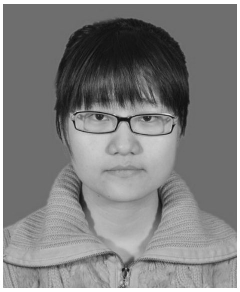
[22] Guo, M.—Ammar, M. H.—Zegura, E. F.: Selecting Among Replicated Batching Video-on-Demand Servers. Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '02), 2002, pp. 155–163, doi: 10.1145/507670.507692.

[23] Chang, C. W.—Huang, G.—Lin, B. et al.: LEISURE: Load-Balanced Network-Wide Traffic Measurement and Monitor Placement. IEEE Transactions on Parallel and Distributed Systems, Vol. 26, 2015, No. 4, pp. 1059–1070.

[24] Kim, H.-Y.: An Energy-Efficient Load Balancing Scheme to Extend Lifetime in Wireless Sensor Networks. Cluster Computing, Vol. 19, 2006, No. 1, pp. 279–283.

[25] Bui, H.—Johnson, A.—Jacob, R. et al.: Multipath Load Balancing for $M \times N$ Communication Patterns on the Blue Gene/Q Supercomputer Interconnection Network. 2015 IEEE International Conference on Cluster Computing (CLUSTER 2015), 2015, pp. 833–840, doi: 10.1109/CLUSTER.2015.140.

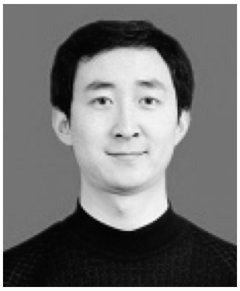
[26] Simchi-Levi, D.—Trick, M. A.: Introduction to Little's Law as Viewed on Its 50th Anniversary. Operations Research, Vol. 59, 2011, No. 3, p. 535.



Pingting Hao received her B.E. degree in 2013 from Jilin University. She is currently pursuing her Ph.D. in the Department of Computer Science, Jilin University, China. Her research interests include distributed computing, content delivery networks and software defined networks.

Liang Hu received his B.Sc. degree from the Harbin Institute of Technology (HIT), Harbin, and M.Sc. and Ph.D. degrees from the College of Computer Science and Technology, Jilin University (JLU), Changchun, China. He has been Professor since 2002 and Ph.D. supervisor since 2003 with the School of Jilin University. His current research interests include distributed computing, network computing and security, data security and privacy.

Jingyan Jiang received her B.E. degree in 2012 from Jilin University. She is currently pursuing her Ph.D. in the Department of Computer Science, Jilin University, China. Her research interests include distributed computing, multimedia networks and software defined networks.



Xilong Che received his M.Sc. and Ph.D. degrees in computer science from Jilin University, in 2006 and 2009, respectively. Currently, he is Associate Professor and Master Supervisor at the College of Computer Science and Technology, Jilin University, China. His current research areas are parallel and distributed computing, machine learning, and related applications. He is also a member of the IEEE and the corresponding author of this paper.