Addressing the Out-of-date Problem for Efficient Load Balancing Algorithm in P2P Systems

Khaled Ragab and Moawia Elfaki Yahia

Abstract Load-balancing is of major significance for large-scale decentralized networks such as Peer-to-Peer (P2P networks in terms of enhanced scalability and performance. P2P networks are considered to be the most important development for content distribution and sharing infrastructures. Load balancing among peers in P2P networks is critical and a key challenge. This paper addresses the out-of-date problem as a result of node's state changes during loads movement among nodes. Consequently, this work proposes a load balancing algorithm that is based on extensive stochastic analysis and virtual server concept in P2P System. Finally, this work is complemented with extensive simulations and experiments.

Keywords Out-of-date problem · Peer-to-peer networks · Virtual servers · Load balancing

1 Introduction

Recently **Peer-to-Peer (P2P)** paradigm is an increasingly popular approach for developing various decentralized systems especially the internet applications. P2P is a class of applications that takes advantage of resources e.g. storage, cycles, content, human presence, available at the edges of the Internet [1]. P2P systems

K. Ragab (✉)
Computer Science Division, Mathematics Department, College of Science, Ain Shams University, Cairo, Egypt
e-mail: kabdultawab@kfu.edu.sa

M. E. Yahia
Computer Science Department, College of Computer Science and Information Technology, Hofuf, Saudi Arabia

K. J. Kim and K.-Y. Chung (eds.), *IT Convergence and Security 2012*,
Lecture Notes in Electrical Engineering 215, DOI: 10.1007/978-94-007-5860-5_56,

offer an alternative to such traditional client–server systems for several application domains. They have emerged as an interesting solution for sharing and locating resources over the Internet. Moreover, P2P systems do not have a single point of failure and can easily scale by adding further computing resources. They are seen as economical as well as practical solutions in distributed computing. In P2P systems, every node (peer) of the system acts as both client and server (**servant**) and provides part of the overall resources/information available from the system. Each node often has different resource capabilities (e.g. processor, storage, and bandwidth) [2]. Thus, it is required that each node has a load proportional to its resources capabilities. On account of the dynamism natures of the P2P systems, it is difficult to ensure that the load is uniformly distributed across the system. In particular, this paper considers a P2P system of M nodes in which nodes join/leave, and data entity inserted/deleted continuously. Similarly to [3–5] this paper assumes node and data entity have been assigned identifiers that chosen randomly. Thus, there is a $\Theta(log\ M)$ imbalance factor in the number of data entities stored at a node. Additionally, the imbalance factor becomes more worse if the P2P applications associate semantics with data entity IDs since IDs will not be uniformly distributed.

Consequently, it is important to design mechanisms that balance the system load. There are two distinct strategies to distribute the system workload [6]. First, load balancing algorithms that strive to equalize the workload among nodes. Second, load sharing algorithms which simply attempt to assure that no node is idle while jobs at other nodes are waiting for service. Load balancing techniques in P2P systems should be scalable and cope with its large size. They should place or re-place shared data entities optimally among nodes while maintaining an efficient overlay routing tables to redirect queries to the right node.

The communication delays among peers significantly alter the expected performance of the load balancing schemes. Due to such delay, the information that a particular peer has about other peers at any time is dated and may not accurately represent the current state of the other peers. For the same reason, a load sent to a recipient peer arrives at a delayed instant. In the mean time, however, the load state of the recipient peer may have considerably changed from what was known to the transmitting peer at the time of load transfer. This paper proposes a stochastic dynamic load balancing algorithm that tackles the out-of-date problem.

The remainder of this paper is organized as follows. Section 2 introduces a survey for the load balancing algorithms. Section 3 exposes the proposed stochastic load balancing model and algorithm. Evaluation of the proposed has been discussed in Sect. 4. Section 5 draws a conclusion of this paper.

2 Load Balancing Survey

Load balancing is the problem of mapping and remapping workload in the distributed system.

2.1 Load Balancing Design

Load balancing design determines how nodes communicate and migrate loads for the purpose of load balancing. It moves workload from heavily loaded nodes (senders) to lightly loaded nodes (receivers) to improve the system overall performance [15]. Load balancing design includes four components that can be classified as follows [16, 17].

- *Transfer policy*: It decides whether a node is in a suitable state to participate in a load transfer; either receiver or sender.
- *Location policy*: Once the transfer policy decides that a node is a receiver or sender. The location policy takes the responsibility to find a suitable sender or receiver.
- *Selection policy*: Once the transfer policy decides that a node is a sender, the selection policy specifies which load should be transferred. It should take into account several factors such as load transfer cost, and life time of the process that should be larger than load transfer time.
- *Information policy*: It decides when and how to collect system state information.

Load balancing designs are categorized into static and dynamic. With a static load balancing scheme, loads are scattered from sender to receiver through deterministic splits. Static schemes are simple to implement and easy to achieve with slight overhead [13]. They perform perfectly in homogenous systems, where all nodes are almost the same, and all loads are same as well. On the other hand, the dynamic load balancing schemes make decisions based on the current status information [15]. Accordingly, the transfer policy at certain node decides to be a sender or receiver, the selection policy selects the load to be transferred. The dynamic load balancing schemes perform efficiently when its nodes have heterogeneous loads, and resources. The typical architectures of dynamic load balancing schemes can be classified into centralized, distributed, and topological. In a centralized scheme, a central server "*coordinator*" receives load report from the other nodes, while overloaded nodes request the coordinator to find underloaded nodes [17]. In distributed architecture, each node has a global or partial view of the system status. Consequently, the transfer policy at each node can locally decide to transfer a load either out from it (*sender-initiated*) or into it (*receiver-initiated*) [18]. Then, the location policy at each node probes a limited number of nodes to find a suitable receiver or sender. *Kruger* [19], proposed symmetrically-initiated adaptive location policy that uses information gathered during its previous searches in order to keep track of the recent state of each node in the system. It finds a suitable receiver when a heavily-loaded node wishes to send a load out, and finds a suitable sender when a lightly-loaded node wishes to receive a load. Finally, in a system with large number of nodes, a topological scheme should be used [20]. It partitions nodes into groups. The load balance is performed in each group first, then, a global load balance among groups will be performed. However nodes in the

hierarchical architecture [21] are organized into a tree. Inner nodes gather the status information of its sub-trees. Then, load balancing is performed the leaves to the roots of the tree.

2.2 P2P Load Balancing

Load balancing is a critical issue for the efficient operation of the P2P systems. Recently, it attracted much attention in the research community especially in the *distributed hashed table* (*DHT*) based P2P systems. Namespace balancing is struggling to balance the load across nodes by ensuring that each node is responsible for a balanced namespace. This is valid only under the assumption of uniform workload and uniform node capacity. Otherwise, there is a $\Theta(log\ M)$ imbalance factor in the number of objects stored at a node. To mitigate this imbalance two categories, *node placement* and *object-placement* load balancing techniques were proposed. In the node placement technique, nodes can be placed or replaced in locations with heavy loads. For example, a node in the *Mercury* load balancing mechanisms [7] is able to detect a lightly loaded range, and move there if it is overloaded. In object placement technique, objects are placed at lightly loaded nodes either when they are inserted into the system [11] or through dynamic load balancing schemes based on the virtual servers (*VSs*) concept [8], whose explicit definition and use for load balance was proposed by *Godfrey* et. al. [9] and Rao et. al. [11]. In [8], a virtual server represents a peer in the *DHT*; that is the storage of data items. In addition, routing takes place at the virtual server level rather than at the physical node level. In this paper, we assume virtual server as a virtual machine that is able to process a set of jobs like physical machine. Each physical node creates a pool of *VSs* as seen in Fig. 1. Load balancing could be achieved by migrating *VSs* from heavily loaded physical node to lightly loaded physical node. One main advantage of using *VSs* for balancing the load is that approach does not require any changes to the underlying network. In fact, the

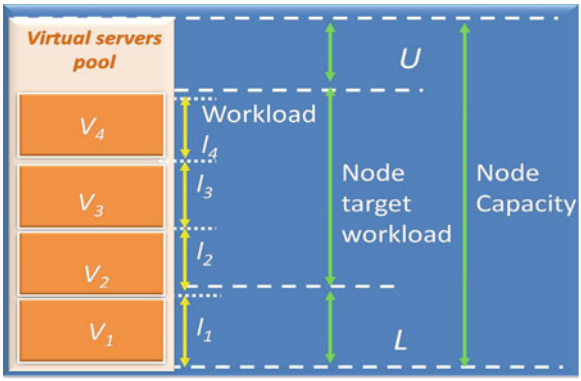

Fig. 1 Node's load specification

transfer of a virtual server can be implemented simply as a peer leaving and peer joining the system. In [11], *Rao* et. al. proposed three simple and static load balancing schemes: "*one-to-one*", "*one-to-many*" and "*many-to-many*". Godfrey et. al. combines both "*one-to-many*" and "*many-to-many*" schemes and uses them in different scenarios [9]. Clustered *VSs* scheme is presented in [12] that optimized the basic *VS* framework to reduce the overhead involved in the *VS* framework. However, *VSs* cannot be moved, and therefore, the scheme cannot respond to dynamic changes in network conditions.

This paper focuses on the design and analysis of P2P load balancing algorithm based on stochastic analysis [23, 24] and based on the *VSs* concept [8].

2.3 Challenges: P2P Load Balancing

Load balancing techniques in P2P systems are facing challenges coming from the characteristics of these systems. First, the size of the P2P system is large that means a scalable load balancing technique is required. Second, dissimilar to the traditional systems, nodes of a P2P system are not replicas and requests cannot be executed in any node. If nodes have dated, inaccurate information about the state of other nodes, due to random communication delays between nodes, then this could result in unnecessary periodic exchange of loads among them. For example, an overloaded node removes some of its virtual servers. However, such simple deletion will cause the problem of "*load thrashing*[1]", for the removed virtual servers may make other nodes overloaded. Consequently, this paper proposes a stochastic P2P load balancing algorithm that approximately determines the minimum amount of time to change the node's state from overloaded to underloaded and vice versa. Comparing that time with the required time to migrate virtual servers enable us to come to a careful decision. Accordingly, the proposed algorithm undoubtedly avoids the load thrashing. To the best of the author's knowledge, there is no any load balancing algorithm for the P2P system based upon the following stochastic analysis.

3 Load Sharing Algorithm

3.1 Model

This paper considers a P2P system consisting of M physical nodes (peers), denoted by P_i, $1 \leq i \leq M$. Each peer can be modeled as a queuing system, such as M/M/1,

[1] *Load thrashing* is a condition when the load balancing algorithm is engaged in moving virtual servers back and forth between nodes. .

M/D/1, etc. Each physical node P_i has a capacity C_i that corresponds to the maximum amount of load that it can process per unit of time. Nodes create virtual servers (*VSs*), which join the P2P network. Therefore, it can own multiple non-contiguous portions of the DHT's identifier space. Each virtual server participates in the DHT as a single entry (e.g. routing table). Moreover, each virtual server stores data items whose IDs fall into its responsible region of the DHT's identifier space. As seen in Fig. 1, a node P_i might have n *VSs* $v_1, v_2, \ldots, v_n$; where $n = VSset.size$. Each v_j has load l_j; (for j = 1, ..n). The load of peer P_i in a unit of time is $L_i = l_1 + l_2 + \ldots + l_n$. The utilization of a node's P_i is L_i/C_i. From the perspective of load balancing, a virtual server represents certain amount of load (e.g. the load generated by serving the requests of the data items whose IDs fall into its responsible region) [25]. To avoid fluctuations in workload nodes should operate below their capacity. If a node finds itself receiving more load L_i than the upper target load U (*i.e.* $(L_i/C_i) > U$), it considers itself overloaded. A node P_i also has load L_i less than L is considered to be underloaded. An overloaded node is not able to store objects given to it route packets, or carry out computation, depending on the application.

Definition 1 A node P_i is in one of the following state as follows

$$S_i = \begin{cases} Underloaded & if\ Q_i < L \\ Normal & if\ L \leq Q_i \leq U \\ Overloaded & if\ U < Q_i \end{cases}$$

Clearly the state space Q_i consists of non-negative integers sub-divided into three disjoint regions [0, L), [L,U], and (U, ∞) corresponding to underloaded, normal, and overloaded state respectively.

A P2P system is defined to be balanced if the sum of the load L_i of a physical node P_i is smaller than or equal to the target load of the node for every node P_i, $1 \leq i \leq M$ in the system. When the system is imbalanced, the goal of a load balancing algorithm is to move *VSs* from overloaded node to underloaded one with minimum load transfer overheads.

The amount of overload to be transferred from the overloaded node P_i; $1 \leq i \leq M$ is a random variable denoted by A is given by

$$A(p_i) = \max(0, Q_i - U\} = \begin{cases} Q_i - U & if\ Q_i > U \\ 0 & Otherwise \end{cases}$$

Similarly, the amount of underload that can be accepted at the underloaded peer P_i; $1 \leq i \leq M$ is a random variable denoted by B is given by

$$B(p_i) = \max(0, L - Q_i\} = \begin{cases} L - Q_i & if\ Q_i < L \\ 0 & Otherwise \end{cases}$$

Definition 2 Let $\{Q(t); t \geq 0\}$ be a stationary[2] stochastic process with state space consisting of non-negative integers. Let S_i and S_j be two distinct non-negative numbers. The First Passage Time (FPT) between states S_i and S_j is denoted by FPT(S_i,S_j), is given by

$$FPT(S_i, S_j) = \begin{cases} \inf\{t; \quad Q(t) = S_j, Q(0) = S_i)\} & \text{if } S_i \neq S_j \\ 0 & \text{if } S_i = S \end{cases}$$

It is a random variable which measures the minimum amount of time needed to reach state S_j from state S_i. We note that because the same stochastic process $Q(t)$ is stationary, translating the above events by a fixed amount of time has no effect upon the probability distribution of $FPT(S_i,S_j)$. In fact, a first passage time from state S_i to state S_j can be divided into two parts, namely the first transition out of state S_i (say S_k) followed by the first passage from S_k to S_j.

Assume that, $i < j$; since changes of state have unit magnitude in a birth and death of load, then

$$FPT_{ij} = FPT_{ik} + FPT_{kj}\ i<k<j;\ k = i+1, i+2, \ldots, j-1 \text{ thus}$$

$$FPT_{ij} = \sum_{k=i}^{j-1} FPT_{k,k+1} \qquad i<j \tag{1}$$

Similarly, if i > j

$$FPT_{ij} = \sum_{k=j}^{i-1} FPT_{k+1,k} \qquad i>j \tag{2}$$

Let $H_{ij}(t) = P\{FPT_{ij} \leq t\}$ and considering that the summands in both Eqs. (1) and (2) are independent. Thus, if we apply *Laplace* transformer $\tilde{H}_{ij} = \prod_{k=i}^{j-1} \tilde{H}_{k,k+1}(s)$ for $H_{ij}(t)$, then we can show that:

$$\tilde{H}_{ij}(s) = \prod_{k=i}^{j-1} \tilde{H}(s,k,k+1) \qquad ;i<j \qquad (Upward) \tag{3}$$

$$\tilde{H}_{ij}(s) = \prod_{k=j}^{i-1} \tilde{H}(s,k+1,k) \qquad ;i>j \quad (Downward) \tag{4}$$

[2] A **stationary stochastic process** has the property that the joint distribution don not depend on the time origin. The stochastic process $\{Q(t); t\in\Im\}$ is called stationary if $t_i\in\Im$ and $t_i + s \in\Im$, $i = 1, 2, \ldots, k$ (k is any positive integer), then $\{Q(t_1), \ldots, Q(t_k)\}$ and $\{Q(t_1 + s), \ldots, Q(t_k + s)\}$ have the same joint distribution [22].

Clearly the distribution of the first passage time of unit downward is independent of starting state while the distribution of the first passage time of unit upward depends upon starting state.

3.2 Load Sharing Edge

The aim of this section is to study the *FPT* of the transition from normal state to overloaded, overloaded to normal, underloaded to normal, etc. For each transfer pair, *FPT* will be computed to predicate the future behavior of the transfer pair before the load transfer (i.e. virtual server migration) decision is taken.

Definition 3 Let *[Q(t), R(t)]* be a transfer pair with $Q(t) = X$ *and* $R(t) = Y$, where $X > U$ and $Y < L$. **The Load Sharing Edge (LSE)** between Q and R is a random variable *E(X,Y)* which is defined as follows:

$$E(X, Y) = min\{FPT(X, U, FPT)(Y, L)\}$$

Where, *FPT(X,U)* is the first passage time to move from state X to state U and *FPT(Y,L)* is the first passage time to move from state Y to state L.

LSE is the period of time within which the overloaded node must complete transferring load to the underloaded node before the overloaded node identifies that is unnecessary to transfer load or the underloaded node becomes ineligible to receive a transferred load. Assume the load transfer time is denoted by Δ. It is the time needed to package and send the load (i.e. the least loaded virtual server that will release overload) to sink R. Thus, the load transfer must be initiated only if $LSE > \Delta$. Since *LSE* is a random variable we need to formulate the transfer criterion in terms of probabilities. Assume the probability that *LSE* exceeds Δ is $P\{E(X,Y) > \Delta\}$. Therefore, the load transfer must be initiated if $P\{E(X,Y) > \Delta\}$ is large. These considerations led to the formulation of a class of rules so called *Quantile rules*. The quantile of a probability distribution function is defined as follows:

Definition 4 Let $F(t),\ t \geq 0$ be the probability distribution function of a non-negative random variable X. Let $0 < \beta < 1$. The β-quantile of F is a non-negative real number q_β satisfying

$$1 - F(q_\beta) = \beta, \quad P\{X \geq q_\beta\} = \beta.$$

From Definition 4, the β-quantile rule for load transfer was introduced.

Definition 5 Given a transfer pair *[Q(t), R(t)]* and a load transfer time is Δ. Also, for $0 < \beta < 1$, let q_β be the β-quantile of the probability distribution of the LSE between *Q(t)* and *R(t)*. Then the load transfer is initiated only if $q_\beta > \Delta$.

The proposed algorithm in this paper uses a β-quantile rule before transferring load and ensures that $1 \geq P\{E(X,Y) \geq \Delta\} \geq \beta$. In general β can be taken 0.90 or large.

The probability distribution of the random value *LSE* is given as follows

$$P_e(t;X,Y) = P\{E(X,Y) \leq t\} \text{ for } \quad t \geq 0.$$

Thus, the probability distribution function of the load sharing edge *LSE* between pair *Q(t)* and *R(t)* is given by

$$P_e(t;X,Y) = \ 1 - [1 - F(t;X,U)] \times [1 - G(t;Y,L)].$$

Where *F(t; X,U) and G(t; Y,L)* are the probability distribution of the first passage time from X to U and from Y to L in the queues {Q(t)}, {R(t)}, respectively. Each node is modeled as *M/M/1* queue, in which processes arrive according to a Poisson process with mean arrival rate λ, then processed with exponential service time [23, 24] and with mean service rate μ.

Lemma 1 *Assume constant birth rates* $\lambda = \lambda_0 = \lambda_1 = \ldots$ *and death rates* $\mu = \mu_0 = \mu_1 = \ldots$, *then the probability distribution function is*

$$H_{k+1,k}(.) = H_{1,0}(.)\ k = 0, 1, \ldots$$

Proof

$FPT_{k+1,k}$ is the time that elapses before the cumulative number of deaths first exceeds the cumulative number of births when $X(0) = k + 1$. Also, the value of $H_{k+1,k}(.)$ do not depend on *X(0)*, [23, 24]. □

Lemma 2 *The Laplace–Stieljes transform of the probability distribution function of the first passage time from state k to state 0 in an M/M/1 queue is*

$$\tilde{H}_{k,0}(s) = \left[\frac{s + \lambda + \mu - \sqrt{(s + \lambda + \mu)^2 - 4\lambda\mu}}{2\lambda}\right]^k \tag{5}$$

Proof

Assume that, the first passage time FPT_{ij} can be expressed as

$$FPT_{ij} = S_1 + \begin{cases} FPT_{i+1,j} & \text{if } X(s_1) = i + 1 \\ FPT_{i-1,j} & \text{if } X(s_1) = i - 1 \end{cases}$$

Where S_1 is the time of the first transition. Assume that $H_{ij}(t) = P\{FPT_{ij} \leq t\}$, as well FPT_{ij} can be upward or downward after the first transition S_1. Using Theorem 4–7, [23, 24], $H_{ij}(t)$ can be expressed as follows

$$H_{ij}(t) = \lambda_i \int_0^t H_{i+1,j}(t-x)e^{-(\lambda_i+\mu_i)x}dx + \mu_i \int_0^t H_{i-1,j}(t-x)e^{-(\lambda_i+\mu_i)x}dx \quad \text{(i)}$$

Taking *Laplace–Stieljes* transform on both side of Eq. (i) and use the convolution property, then the following equation can be obtained:

$$\tilde{H}_{i,j}(s) = \frac{\lambda_i \tilde{H}_{i+1,j}(s) + \mu_i \tilde{H}_{i-1,j}(s)}{s + \lambda_j + \mu_j} \quad \text{(ii)}$$

Thus,
Set i = 1, and j = 0

$$\tilde{H}_{1,0}(s) = \frac{\lambda_0 \tilde{H}_{2,0}(s) + \mu_0 \tilde{H}_{0,0}(s)}{s + \lambda_0 + \mu_0} \quad \text{(iii)}$$

From lemma 1, $\lambda = \lambda_0$, $\mu = \mu_0$ and from Eq. (4)
$\tilde{H}_{i,j}(s) = \prod_{k=j}^{i-1} \tilde{H}_{k+1,k}(s) \quad ; i > j$. Thus, $\tilde{H}_{2,0}(s) = \tilde{H}_{1,0}(s) \times \tilde{H}_{2,1}(s) = \left(\tilde{H}_{1,0}(s)\right)^2$ from Eq. (iii), we obtain the following quadratic equation

$$\lambda\left[\tilde{H}_{1,0}(s)\right]^2 - (s + \lambda + \mu)\tilde{H}_{1,0}(s) + \mu = 0 \quad \text{(iv)}$$

Equation (iv) has two solutions, we consider the solution which satisfies that $\tilde{H}_{1,0}(s) \leq 1$

$$\tilde{H}_{k,0}(s) = \left[\frac{s + \lambda + \mu - \sqrt{(s+\lambda+\mu)^2 - 4\lambda\mu}}{2\lambda}\right]; \tilde{H}_{1,0}(s) \leq 1, \text{for real.}$$

Hence,

$$\tilde{H}_{k,0}(s) = \left[\tilde{H}_{1,0}(s)\right]^K = \left[\frac{s + \lambda + \mu - \sqrt{(s+\lambda+\mu)^2 - 4\lambda\mu}}{2\lambda}\right]^k. \quad \square$$

Corollary 1 *The density function of the first passage time from state k to state 0 in an M/M/1 queue is*

$$h_{k,o}(t) = ke^{-(\lambda+\mu)t} I_k\left(2t\sqrt{\lambda\mu}\right) \frac{(\mu/\lambda)^{k/2}}{t} \quad (6)$$

For $t > 0$ m; where I_k is the Modified Bessel function of order k.

Proof

If Eq. (5) is bona fide Laplace transform, it is the Laplace transform of h(t), [23, and 24]. From Laplace transform,

$$\wp\{e^{-ct}f(t)\} = f(s+c);\ \text{set}\ (c=\lambda+\mu)\ \text{and}\ w=(s+\lambda+\mu)$$

So we can write Eq. (5) as

$$\tilde{H}_{1,0}(w) = \left[\frac{w-\sqrt{w^2-4\lambda\mu}}{2\lambda}\right]\square$$

While the Laplace transformation $\wp\{I_1(at)/t\} = \frac{s+\sqrt{s^2+a^2}}{a}$, [23] then we can say that the numerator is the Laplace transform of $2(\lambda\mu)^{1/2}I_1(2t(\lambda\mu)^{1/2})/t$; thus, $H_{1,0} = (s)$ is the Laplace transform of Eq. (5). So we have

$$h_{1,o}(t) = ke^{-(\lambda+\mu)t}I_1\left(2t\sqrt{\lambda\mu}\right)\frac{\sqrt{(\mu/\lambda)}}{t}. \qquad \square$$

Hence,

$$h_{k,o}(t) = ke^{-(\lambda+\mu)t}I_k\left(2t\sqrt{\lambda\mu}\right)\frac{(\mu/\lambda)^{k/2}}{t}; t>0$$

Lemma 3 (Downward) *The probability distribution of the first passage time form state k to state 0 in an M/M/1 queue is:*

$$H(x;k,0) = 1.0 - k\mu^k\sum_{n=0}^{\infty}\frac{(\lambda\mu)^n}{(\lambda+\mu)^{2n+k}}\Gamma(2n+k,x) \tag{7}$$

Where $\Gamma(\quad)$*is the incomplete Gama function.*

Proof

Assume that the modified Bessel function of order k is

$$I_k(x) = \left(\frac{x}{2}\right)^k\sum_{n=0}^{\infty}\frac{(x^2/4)^n}{n!(n+k)!}; x\geq 0$$

Set $x = 2t\sqrt{\lambda\mu}$. Thus,

$I_k(2t\sqrt{\lambda\mu}) = \left(\frac{2t\sqrt{\lambda\mu}}{2}\right)^k\sum_{n=0}^{\infty}\frac{(t^2\lambda\mu)^n}{n!(n+k)!}; x\geq 0$ By substitute into Eq. (6), we can compute the density function $h_{k,o}(t)$ as follows:

$$h_{k,o}(t) = k\mu e^{-(\lambda+\mu)t}\sum_{n=0}^{\infty}\frac{(\lambda\mu)^n t^{k+2n-1}}{n!(n+k)!}; t\geq 0$$

From the definition of the probability distribution function, we have $H_{k,0}(x) = \int_0^x h_{k,0}(t)\,dt$ and $H_{k,0}(x) = 1 - \int_x^\infty h_{k,0}(t)\,dt$.

Thus, $H_{k,0}(x) = 1 - \int_x^\infty k\mu^k e^{-(\lambda+\mu)t} \sum_{n=0}^\infty \frac{(\lambda\mu)^n t^{k+2n-1}}{n!(n+k)!}\,dt$.

If we exchange the infinite sum and the integral we get

$$H_{k,0}(x) = 1 - \sum_{n=0}^\infty \frac{(\lambda\mu)^n}{n!(n+k)!} \int_x^\infty e^{-(\lambda+\mu)t}\, t^{k+2n-1}\,dt\ ;\ \Gamma(i,x) = \int_x^\infty e^{-t} t^{i-1} dt.$$

Hence,

$$H(x;k,0) = 1.0 - k\mu^k \sum_{n=0}^\infty \frac{(\lambda\mu)^n}{(\lambda+\mu)^{2n+k}} \Gamma(2n+k,x). \qquad \square$$

Lemma 4 *The probability density function of the first passage time of the M/M/1 queue from state 0 to state 1 is* $h_{0,1}(t) = \lambda e^{-\lambda t}$.

Proof

Let $\tilde{h}_{i,j}(s) = \frac{\lambda_i \tilde{h}_{i+1,j}(s) + \mu_i \tilde{h}_{i-1,j}(s)}{s+\lambda_j+\mu_j}$ set $i = 0$, $j = 1$ (i.e. there is no service $\mu_0 = 0$) and $\tilde{h}_{01}(s) = \frac{\lambda_0 \tilde{h}_{1,1,j}(s)}{s+\lambda_0}$. For simplicity set $\lambda_0 = \lambda$. Hence $\tilde{h}_{0,1}(s) = \frac{\lambda}{s+\lambda}$; $\tilde{h}_{1,1} = 1$ since $FPT_{11} = 0$. By computing the inverse of the Laplace transformation we get:

$$h_{0,1}(t) = \lambda e^{-\lambda t}. \qquad \square$$

Lemma 5 (Upward) *The probability distribution function of the first passage time of the M/M/1 queue from state i to state j; i < j is*

$$H_{i,j}(t) = 1 + \lambda^{j-i} \sum_{k=1}^{j} C_k e^{(-r_k t)} \tag{8}$$

where r_k; $1 \le k \le j$, *are j distinct roots of the polynomial of degree j defined recursively as:*

$$D(s-1) = 0,\ D(s,0) = 1;\ D(s,1) = s + \lambda$$
$$D(s,j) = (s+\lambda+\mu)D(s,j-1) - \lambda\mu D(s,j-2); j \ge 2$$

also, for $1 \le k \le j$, $\quad C_k = (s+r_k)\frac{D(s,i)}{sD(s,i)}\Big|_{s=-r_k}$

Proof

In lemma 4, we have prove $\tilde{h}_{0,1}(s) = \frac{\lambda}{s+\lambda}$ that is the Laplace transformation of the probability distribution of the first passage time of FPT_{01}. Assume that $\tilde{h}_{i,j}(s) = \frac{\lambda_i \tilde{h}_{i+1,j}(s) + \mu_i \tilde{h}_{i-1,j}(s)}{s+\lambda_j+\mu_j}$. Set $i = k, j = k+1, \quad \lambda_k = \lambda_{k+1} = \lambda, \quad \mu_k = \mu_{k+1} = \mu$

$$\tilde{h}_{k,k+1}(s) = \frac{\lambda\, \tilde{h}_{k+1,k+1}(s) + \mu\, \tilde{h}_{k-1,k+1}(s)}{s+\lambda+\mu} \tag{i}$$

But, $\tilde{h}_{k+1,k+1}(s) = 1$ since $\mathrm{FPT}_{k+1,k+1} = 0$,

$$\tilde{h}_{k-1,k+1}(s) = \tilde{h}_{k-1,k}(s)\, \tilde{h}_{k,k+1}(s) \quad .$$

Hence,

$$\tilde{h}_{k,k+1}(s) = \frac{\lambda \ + \mu\, \tilde{h}_{k-1,k}(s)\, \tilde{h}_{k,k+1}(s)}{s+\lambda+\mu}$$

$$\tilde{h}_{k,k+1}(s) = \frac{\lambda}{s+\lambda+\left[\mu\left(1-\tilde{h}_{k-1,k}(s)\right)\right]} \tag{ii}$$

Using mathematical induction, we can prove that Eq. (ii) is satisfied for all $k \geq 0$. It can be rewritten as the ratio of two functions $N(s,k)$ and $D(s,k)$. These functions can be defined as follows:

$$L(1) = -\lambda, \qquad L(k) = -(\lambda+\mu),\ k \geq 2$$
$$M(k) = \lambda\mu \qquad k \geq 1$$

Thus,

$$N(s,k) = \lambda D(s,k-1),\ k > 1; D(s,0) = 1$$
$$D(s,k) = [s - L(k)] \times D(s,k-1) - M(k) \times D(s,k-2);\ k \geq 1$$

Where $D(s,k)$ is a polynomial of degree $k; k \geq j$.

Thus, Eq. (ii) can be rewritten as follows:

$$\tilde{h}_{k-1,k}(s) = \frac{N(s,k)}{D(s,k)} = \frac{\lambda D(s,k-1)}{s+\lambda+\mu D(s,k-1) - \lambda\mu D(s,k-2)}, k \geq 2$$

But for general transitions from state i to state j: $0 \leq i \leq j$, is

$$\tilde{h}_{i,j}(s) = \frac{\lambda D(s,i)}{D(s,i+1)} \times \frac{\lambda D(s,i+1)}{D(s,i+2)} \times \ldots \times \frac{\lambda D(s,j-2)}{D(s,j-1)} \times \frac{\lambda D(s,j-1)}{D(s,j)}$$

Thus, we can obtain the Laplace transform of the density of the upward transition from state i to state j, $\tilde{h}_{i,j}(s) = \frac{\lambda^{j-i} D(s,i)}{sD(s,j)}$ by canceling the common terms from the denominator and the numerator from the above equation.

From the definition $LT\left[\int_0^x h(t)dt\right](s) = \frac{LT[h](s)}{s}$, thus

$$\tilde{H}_{i,j}(s) = \frac{\lambda^{j-i} D(s,i)}{sD(s,j)}$$

It is a relation function in which the numerator polynomial has degree I while the denominator polynomial has degree $(j + 1)$ and $(i < j)$. Thus, we can expand $\tilde{H}_{i,j}(s)$ into a finite sum of partial fractions as follows: If $D(0,j) = \lambda^j$for all $j \geq 1$ and$\lambda > 0$ then zero cannot be a root of $D(s,j)$; $j \geq 1$ and it can be written in the following form:

$D(s,j) = (s + r_1)(s + r_2)\ldots(s + r_j)$ where r_k for all $k = 1$ to j are roots of $D(s,j).\tilde{H}_{i,j}(s) = \lambda^{j-i}\left(\frac{C_0}{s} + \sum_{k=1}^{j} \frac{C_k}{(s+r_k)}\right)$ Hence, it can be written as follows:

where $C_k = (s + r_k)\frac{D(s,i-1)}{sD(s,j)}\Big|_{s=-r_k}$; $1 \leq j \leq k$ and

$$C_0 = \frac{D(0,i)}{D(0,j)} = \frac{\lambda^i}{\lambda^j} = \lambda^{i-j}$$

$$\tilde{H}_{i,j}(s) = \frac{1}{s} + \lambda^{j-i} \sum_{k=1}^{j} \frac{C_k}{(s + r_k)}$$

Accordingly, we can invert the Laplace Transform of the above equation. But, each term in the right hand side is in the form $\frac{\alpha}{(s+\beta)}\alpha, \beta$ Where α, β are constants. Each term has inverse Laplace transformation $\alpha e^{\beta t}$. Hence,

$$H_{i,j}(t) = 1 + \lambda^{j-i} \sum_{k=1}^{j} C_k e^{(-r_k t)}$$

Theorem *Let [Q(t), R(t)] be a transfer pair that consists of M/M/1 queues. Let m be the amount of overload and n the amount of underload. Then the probability distribution function of the Load Sharing Edge (LSE)is*

$P_e(t; m, n) = 1 - m\lambda^n \mu^m \left[\sum_{k=1}^{L} C_k e^{-r_k t}\right] \times \left[\sum_{k=0}^{\infty} \frac{(\lambda\mu)^k}{(\lambda+\mu)^{2k+m}} \Gamma(2n + m, x)\right]$ *where,* $r_k, 1 \leq k \leq L$*; are the roots of the polynomial defined recursively as*

$$D(s,-1) = 0;\ D(s,0) = 1;\ D(s,1) = s + \lambda$$
$$D(s,L) = (s + \lambda + \mu)D(s,L-1) - \lambda\mu D(s,L-2);\ L \geq 2;\ 1 \leq k \leq L$$

Also, for

$$1 \leq k \leq L \quad C_k = (s + r_k)\frac{D(s,L-n)}{sD(s,L)}\Big|_{s=-r_k}$$

Proof

The transfer pair has the probability distribution function $P_e(t;i,j)$ of the *LSE*; $i > U, j < L$ which is defined by

$P_e(t;i,j) = 1 - [1 - F(t;i,U)]] * [1 - G(t;j,L)]$ where m is the amount of overload ($m = i\text{-}U$), and n is the amount of underload ($n = L\text{-}j$) then

$P_e(t;m,n) = 1 - [1 - F(t;m,0)]] * [1 - G(t;L-n,L)]$for an M/M/1 queue case *F(t,m,0)* and *G(t, L-n, L)* have been derived from 3 and 5 respectively. Hence,

$$P_e(t;m,n) = 1 - m\lambda^n\mu^m\left[\sum_{k=1}^{L} C_k e^{-r_k t}\right] \times \left[\sum_{k=0}^{\infty} \frac{(\lambda\mu)^k}{(\lambda+\mu)^{2k+m}}\Gamma(2n+m,x)\right].\square$$

Due to the infinite number of terms in the probability distribution $P_e(t;m,n)$of the *LSE* in M/M/1, the following lemma will drive a formula for *LSE* as finite number of terms as follows.

Lemma 6 *For a transfer pair [Q(t), R(t)] with an amount of overload m and an amount of underload n, the Mean Load Sharing Edge MLSE(m, n) is*

$$MLSE(m,n) = -\lambda^n \sum_{k=1}^{L} C_k\left(\frac{1}{r_k} - \tilde{F}(r_k,m,0)\right)$$

where $r_1, r_2, \ldots, r_k, C_k$ are constants defined in the previous theorem. Also,

$$\tilde{F}_{m,0}(s) = \left[\frac{s+\lambda+\mu-\sqrt{(s+\lambda+\mu)^2-4\lambda\mu}}{2\lambda}\right]^m$$

Proof

Since $P_e(t;m,n) = 1 - [1 - F(t;m,0)] \times [1 - G(t;L-n,L)]$ we obtain,

$$MLSE(m,n) = \int_0^\infty P_e(t;m,n)\,dt \quad = \int_0^\infty [1 - F(t;m,0)] \times [1 - G(t;L-n,L)]\,dt$$

From lemma 5, we get

$$MLSE(m,n) = -\int_0^\infty \left[\lambda^n \sum_{k=1}^{L} C_k e^{-r_k t}\right] \times [1 - F(t;m,0)]\,dt$$

But the Laplace transform $LT[q](s)|_{S=0} = \int_0^\infty q(t)\,dt$ and $LT[e^{-\alpha t}q](s)|_{s=0} = LT[q](s+\alpha)$. Set $\alpha = r_k$, *then*

$MLSE(m,n) = -\lambda^n \sum_{k=1}^{L} C_k\left(\frac{1}{r_k} - \tilde{F}(r_k,m,0)\right)$

Consequently, it has been observed that for queuing models in which job arrival and processing rates are independent of queue size, such as *M/M/1* queues. The distribution of *LSE* depends only on the amounts of underload and overload. The following algorithm will use the numeric value of the given mean load sharing edge formula that is based upon the following parameters $(m, n, \lambda_i, \mu_i, \lambda_j, \mu_j, L, q_\beta)$.

3.3 Algorithm

This section introduces the proposed load balancing algorithm based upon the above analysis. Periodically every T seconds, each overloaded peer transfers the exceeds load to the underlaoded peers (i.e. sender-initiated algorithm). This algorithm imposes a β-quantile rule for transferring load. For each pair (λ, μ) a corresponding β-quantile should be determined while β must be taken 0.90 or large. The algorithm is shown in the following scenario:

1. The overloaded node S_i creates a suitable domain (group) D_i from neighbor nodes to the peer S_i. Each node blongs to D_i satisfies $D_i = \{S_j;\ P(FPT_i > -t_{ij}) \geq 0.90 \text{ and } i \neq j\}$. Where t_{ij} is the required communication delay to send a message form node S_i to S_j plus the required time to reply with load transfer of certain virtual server from S_i to S_j. Also, FPT_i is the first passage time of node S_i to tranfer from overloaded state to normal or underloaded state. D_i is an ordered set with respect to the communication t_{ij}. It is implemented as an order linked list.
2. Thus, S_i sends a broadcast messages to all nodes belonging to the doamin D_i. Node S_i must receive a reply from all nodes belonging to D_i within the FPT_i time.
3. Node S_i selects an underlaoded node $S_j \in D_i$ where the mean load sharing edge *MLSE* between S_i and S_j. if $q_\beta > \Delta$ then transfers load (virtual server) from S_i to S_j. Where pair λ and μ are given, Δ is the time needed to transfer load less than or equal to $A(S_j)$, β is 0.90 or large.
4. Repeate step 3 for each underloaded node S_j belonging to D_i whenever FPT_i period doesn't run out yet.

```
Load_balance(Si, T)
{// Every period T seconds Si checks its load
 // jumps above upper load U.
// It should do the following.
Create(Di,Si); // create domain of neighbors
While ( qβ >0) do
  {// repeat the following within a period qβ
      Select Sj∈Di ; // select from order set Di
      Di = Di \ {Si}
      If ( qβ > Δ) then transfer_load(Si, Sj);  }
}
transfer_load (Si, Sj)
{ If  !(Overloaded) then return; //Sender-initiated
 If (Si →VSset.size >1) then
   Choose v∈Si →VSset such that:
    a. Transfer v to Sj will not overlaod Sj
    b. v is the least loaded virtual server that will release
       overload.
    Failing that, let v be the most loaded VS.
    Return the virtual servers reassignment.
    }
```

4 Evaluation

This paper implements an event-based simulation to evaluate the proposed load balancing algorithm. It uses several parameters as follows: default number of virtual servers per node (12), number of nodes (4096), system utilization (0.8), Object arrival rate (Poisson with mean arrival time 0.01 s), average number of objects (1 million), and periodic load balancing period (T = 60 s). This simulation evaluates the following metric. Load Movement Ratio (*LMR*), defined as the total movement cost incurred due to load balancing divided by the total cost of moving all objects in the system at once. In case the value of the *LMR* is 0.1, it infers that the balancer consumes about 10 % of its bandwidth to insert objects. The node arrival rate is modeled by a Poisson process, and the lifetime of a node is drawn from an exponential distribution. This simulation ran with two inter-arrival times 10 and 60 s, it fixes the steady-state number of nodes in the system to 4096 nodes.

Figure 2 plots the LMR metric as a function of system utilization, to study the load moved by the proposed load balancing algorithm as a fraction of the load moved by the underlying DHT due to node arrivals and departures. Figure 2 demonstrates that the load moved by the proposed load balancing algorithm is significantly smaller than the load moved by the underlying DHT especially for small system utilization. In addition, Fig. 2 shows that the LMR with node inter-arrival time 10 s is larger than with node inter-arrival time 60 s. Figure 3 verifies

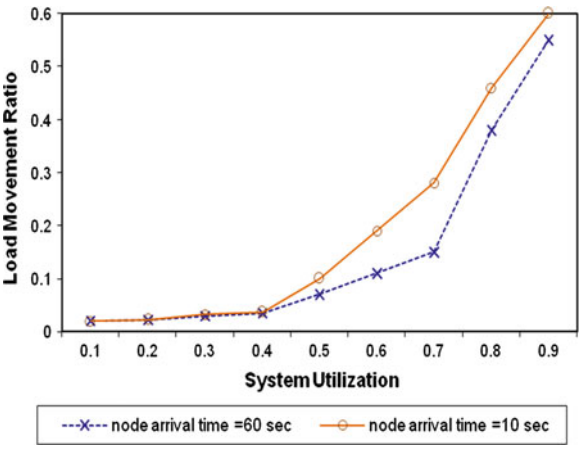

Fig. 2 *LMR* versus system utilization with two node arrival times

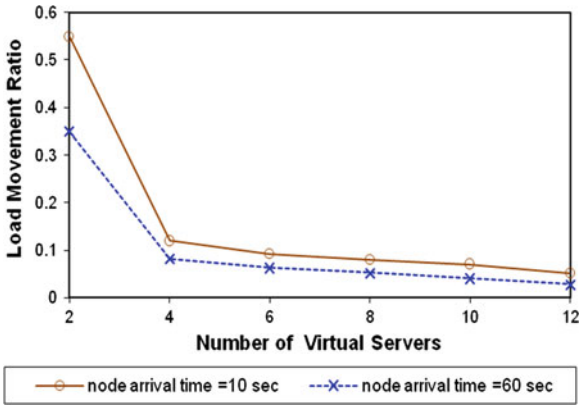

Fig. 3 LMR versus number of virtual servers with two node arrival

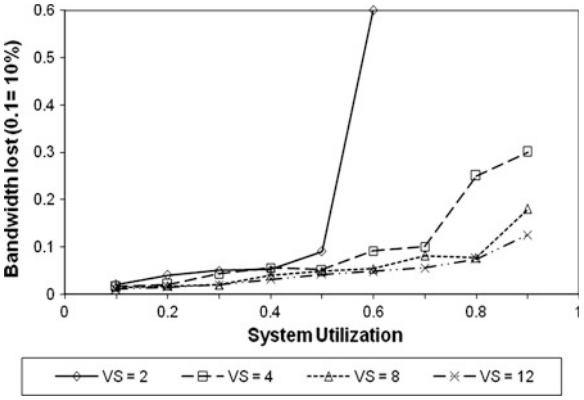

Fig. 4 Bandwidth lost versus system utilization with different number of virtual servers per node

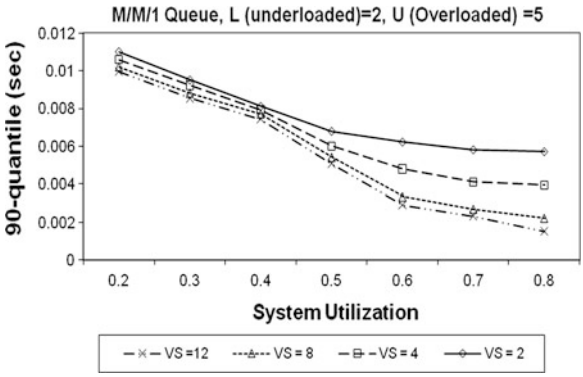

Fig. 5 90-quantile of the LSE versus system utilization with different number of virtual server per node

the perception that increasing the number of virtual servers decreases considerably the fraction of load moved by the underlying DHT. Figure 4 demonstrates that increasing number of virtual servers per node assists load balance at high system utilizations and grants efficient load movements due to low bandwidth lost. Figure 5 plots the 90-quantile of the load sharing edge (LSE) with system utilization when overload is 5 at source node and underload is 2 at the destination node. It demonstrates that the 90-quantile of the LSE tends to be smaller as system utilization increases. As seen from Fig. 5, the 90-quantile of the LSE is 9.94 ms thus load can be transferred only if $\Delta < 9.94$ ms. In addition, increasing the number of virtual servers reduces significantly the 90-quantile that helps in avoiding load thrashing.

In this paper we compare our results with simple load balancing algorithms such as *Random/Round Robin* (*R3*) *load distribution algorithm* [26]. The *R3* algorithm pushes load from an overloaded virtual server to a randomly or in a round robin fashion chosen neighbor that may absorb that load if it has the capacity, or pushes the load further on to another virtual server chosen in the same fashion. The advantages of the *R3* algorithm in compare with the proposed algorithm are as follows: simplicity and statelessness. However, the disadvantages of the *R3* algorithm are as follows: unpredictability and insufficient (random)

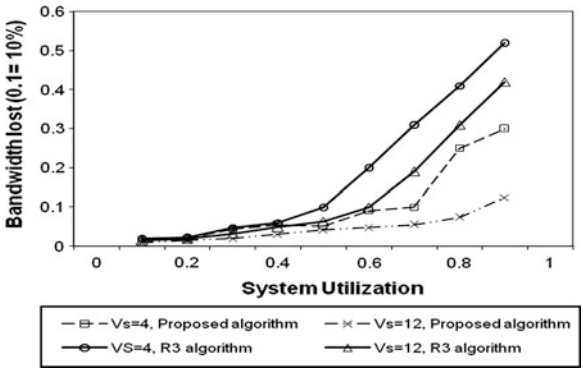

Fig. 6 Compare proposed algorithm with R3 algorithm

convergence on the chance for load thrashing. Figure 6 shows that the proposed algorithm is more efficient than the *R3* algorithm due to load thrashing in R3 that increases the bandwidth lost.

5 Conclusion

Load balancing among peers is critical and a key challenge in peer-to-peer systems. This paper demonstrates a stochastic analysis that avoids the load thrashing and tackles the out-of-date problems due to peer's state changes during load movement (virtual servers migration). Then, it proposes a load balancing algorithm based on the stochastic analysis. An efficient simulation has been carried out that demonstrates the effectiveness of the proposed load balancing algorithm. Further research is to design a P2P load-balancing algorithm based on fuzzy logic control

References

1. Shirky C (2000) Modern P2P definition. http://wwwopenp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html
2. Saroiu S et al (2002) A measurement study of peer-to-peer sharing systems. In: Proceedings multimedia computing and networking conf (MMCN)
3. Sotica I et al (2001) Chord: a scalable peer-to-peer lookup service for internet applications. In: ACM SIGCOMM'01 pp 149–160
4. Rowstron A, Druschel P (2001) Pastry: Scalable distributed object location and routing for large-scale peer-to-peer systems. In Proceedings middleware
5. Ratnasamy S et al (2001) A scalable content- addressable network. In Proceedings ACM SIGCOMM'01, California
6. Derk I et al (1986) Adaptive load sharing in homogenous distributed systems. IEEE Trans on Soft Eng 12(5)
7. Bharambe AR et al (2004) Mercury: supporting scalable multi-attribute range queries. In Proceedings of the conference on applications, technologies, architectures, and protocols for computer communication. ACM, New York
8. Dabek F et al (2001) Wide-area cooperative storage with CFS. In:Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01), pp 2020-2215
9. Godfrey et al (2004) Load balancing in dynamic structured P2P systems. In: Proceedings IEEE INFOCOM
10. Byers J (2003) Simple load balancing for distributed hash table. In: Proceedings of the second international workshop on peer-to-peer systems (IPTPS'03)
11. Rao A et al (2003) Load balancing in structured P2P systems. In Proceedings Of the second international . Workshop on peer-to-peer systems (IPTPS'03)
12. Godfrey PB, Stoica I (2005) Heterogeneity and load balance in distributed hash table. In: Proceedings IEEE INFOCOM
13. Li J, Kameda H (1994) A decomposition algorithm for optimal static load balancing in Tree hierarchy network configurations. IEEE Trans on parallel and distributed systems, 5(5)

14. Casavant TL, Kuhl JG (1988) A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Trans Softw Eng 14(2):141–154
15. Shivaratri NG et al (1992) Load distributing for locally distributed systems. Computer 25(12):33–44
16. Goscinski A (1991) Distributed operating system: the logical design, Addison-Wesly
17. Zhou S (1988) A Trace-driven simulation study of dynamic load balancing. IEEE Trans Softw Eng 14(9):1327–1341
18. Eager DL et al (1985) A comparison of receiver-initiated and sender-initiated adaptive load sharing. SIGMETRICS Perform Eval Rev 13:2
19. Kruger P, Shivaratri NG (1994) Adaptive location policies for global schedule. IEEE Soft Eng 20:432–443
20. Zhou S et al (1993) Utopia: a load sharing facility for large, heterogeneous distributed computer systems. Softw Pract Experience 1305–1336
21. Dandamudi SP, Lo KC (1997) A hierarchical load sharing policy for distributed systems. In: Proceedings of the 5th International. Workshop on modeling, analysis, and simulation of computer and telecommunications systems, MASCOTS. IEEE CS, Washington, DC
22. Heyman DP (1982) Stochastic models in operation research, vol I. MxGraw-Hill Inc, New York
23. Kobayshi H (1978) Modeling and analysis: an introduction to system performance evaluation methodology. Addison-Wesley
24. Hisashi kobayshi et al (2009) System modeling and analysis: foundations of system performance evaluation. Prentice Hall
25. Zhu Y Hu Y (2005) Efficient proximity-aware load balancing for DHT-based P2P systems. IEEE Trans On Parallel Distributed Syst, 16(4) 349–361
26. Andrzejak A, Graupner S, Kotov V, Trinks H (2007) Algorithms for self-organization and adaptive service placement in dynamic distributed systems. Internet systems and storage laboratory