Decentralized Proactive Resource Allocation for Maximizing Throughput of P2P Grid

Sheng Di[1,*], Cho-Li Wang[1,**],

[*]sdi@cs.hku.hk; tel: (852) 62340776; fax: (852) 25598447

[**]clwang@cs.hku.hk; tel: (852) 28578458; fax: (852) 25598447

Email addresses: `sdi@cs.hku.hk` (Sheng Di), `clwang@cs.hku.hk` (Cho-Li Wang)

URL: `http://www.cs.hku.hk/~sdi` (Sheng Di), `http://www.cs.hku.hk/~clwang` (Cho-Li Wang)

[1]Department of Computer Science, The University of Hong Kong, Hong Kong

Abstract

Peer-to-peer Desktop Grids provide integrated computational resources by leveraging autonomous desktop computers located at the edge of the Internet to offer high computing power. The arbitrary arrival and serving rates of tasks on peers impedes the high throughput in large-scale P2P Grids. We propose a novel autonomous resource allocation scheme, which can maximize the throughput of self-organizing P2P Grid systems. Our design possesses three key features: (1) high adaptability to dynamic environment by proactive and convex-optimal estimation of nodes' volatile states; (2) minimized task migration conflict probability (upper bound can be limited to 2%) of over-utilized nodes individually shifting surplus loads; (3) a load-status conscious gossip protocol for optimizing distributed resource discovery effect. Based on a real-life user's workload and capacity distribution (conforming to Pareto distribution), the simulation results show that our approach could get significantly improved throughput with 23.6%$\sim$47.1% reduction on unprocessed workload compared to other methods. We also observe high scalability of our solution under dynamic peer-churning situations.

Keywords: System Throughput, Fully-decentralized Resource Allocation, Autonomous Decision Conflict, P2P desktop Grid system

1. Introduction

Peer-to-peer Desktop Grids, evolving from two pertinent solutions, i.e., Grids and Peer-to-Peer (P2P) systems, provide a new form of wide-area resource sharing by leveraging autonomous desktop computers located at the edge of the Internet. Such systems are well suited for those willing to quickly deploy a computing grid without requiring any centralized administration. In the past decade, we have witnessed Peer-to-peer Desktop Grids have evolved from the emergence to the stable and production status. Examples include BonjourGrid [1], Condor-Flock P2P [2], P2PGrid [3], Alchemi [4], PastryGrid [5], Self-Gridron [6], Harmony [7], etc. The inherent wide distribution, heterogeneity, and dynamism of Desktop Grids make them well suited to the execution of loosely-coupled parallel applications, such as the Bag-of-Tasks (BoT) applications whose tasks are completely independent on each other. Trends also show that the future P2P Grids will instead be composed of extremely large number of individual machines [8].

In a P2P Grid, each user offers his/her computer to execute the tasks submitted by local users with arbitrary arrival rates or tasks migrated from other peers. Since centralized algorithms may be unrealistic on such large-scale platform, a local scheduler at each peer node is activated periodically to determine if the tasks should be executed locally or at a remote host for dynamic load balancing and better resource utilization.

To achieve high throughput for the execution of applications in a large-scale P2P Grid system is a challenging work because machine's computing power is heterogeneous in nature, the workload injected to the system may vary from time to time, and the resource availability is usually subject to the users' own free will which is uncontrollable. The autonomy in performing task scheduling also makes it difficult to precisely predict the runtime load status and resource availability at each peer node due to the mutual impact of their on-line scheduling decisions. Solutions that can adapt to the heterogeneity and dynamism through dynamic load balancing in a decentralized and scalable manner is a step toward realizing the ideal computing ability. To maximize the use of available resources, we need to solve the imbalanced load distribution problem dynamically and variance of processing capacity among heterogeneous peers, as well as the resource contention problem of their on-line scheduling decisions.

Recently, more and more heuristic strategies have been proposed to maximize the system throughput by making use of load balancing in existing P2P Grid systems [2, 4, 7, 9, 10]. Traditional task scheduling strategies mainly focus on minimizing the system's makespan (i.e. the time required to complete all tasks) for achieving high system throughput [11, 12]. However, makespan minimization is not suitable for large-scale distributed platforms as acquiring all various parameters (e.g., workload of individual tasks and the resource availability of all nodes) might take long time, which is also error-prone. Considering the heterogeneity of computing nodes, many other task scheduling solutions [13, 14, 15, 16, 17] are designed based on a *fairness index* criterion of nodes' workload levels. Unlike the traditional concept of fairness on user tasks' service level (e.g. the fairness of task's response), the fairness of nodes' workload levels is usually used to evaluate the system's load balancing level and guide schedulers to (re)distribute workload among nodes proportional to their computing capacities, achieving more balanced resource utilization. Specifically, Jain's fairness index [18] has been commonly adopted to identify underutilized resources and higher fairness value [1] is usually leveraged to denote more balanced load distribution, which may imply higher system throughput to certain extent [19] as well. For example, T. Repantis et al. [16] try to improve the system performance by maximizing the fairness index of load distribution. Y. Drougas et al. [14] also devised an autonomous algorithm by means of the load distribution fairness, aiming to improve the P2P system's performance. However, there are no theoretical propositions to confirm the relation of system throughput and fairness index of workload level, thus most existing solutions that alleviate unfairness of resource utilization can only be based on experiences and may actually not necessarily achieve high throughput.

In this research, we studied the autonomous resource allocation problem in P2P desktop Grid system with large number of nodes and user tasks. We focus on batch mode scheduling for applications: the jobs arriving at the same period at each peer node are grouped into one batch and are scheduled unilaterally by the local scheduler without coordination. All further arriving jobs after the scheduler is activated are delayed to be scheduled in the next period. Via dynamically determining the location for executing the submitted tasks, our goal is to maximize system-wide throughput.

This work possesses three contributions:

- We formally prove that maximizing Jain's fairness index of node's workload level is not a sufficient condition for achieving high system throughput. We derive a new metric, called *average load level* (*ALL*) (i.e., the mean value of nodes' load levels) [2], which could accurately reflect the system throughput

[1]usually at the range of [0,1]. The fairness index is 1 when all peers are equally loaded.

[2]a node's load level is defined as its total workload of all tasks assigned to it divided by its

from the statistical view. We prove that the accuracy of *average load level* will not be impacted by the distribution of node capacities in a large-scale heterogeneous system with vast nodes and tasks.

- We propose a convex-optimization analysis based on a proactive load prediction model to determine the optimal load amount for migration. The method overcomes the slow resource state propagation problem generally encountered in a large-scale P2P network [14, 11, 20], which tried to aggregate global or partial resource states via various gossip protocols [21, 22] and adapt to fast changing task arrival rates that may cause delayed or inaccurate load migration decisions.

- We propose a fully-decentralized task scheduling algorithm that can mitigate decision-conflict among autonomous schedulers. Our solution can avoid the undesired situation where some under-loaded peers suddenly become heavily loaded due to too many tasks migrated from outside. We theoretically derive an upper bound for the decision-conflict probability by leveraging a stochastic theory - Bernoulli trial model, where the selection probability of target nodes is proportional to their relative load status with respect to the average load level. Although the stochastic theory has been widely used in other P2P systems like chunk scheduling of P2P video streaming [23, 24], it is rarely used in the P2P Grid model because the tasks in Grid environment are heterogeneous with different workloads and cannot be split or replicated as flexibly as video chunks.

computing capacity, or intuitively explained as the expected running time to complete all the tasks in its task queue

We build an emulated WAN testbed using Brite tool [25] and perform our simulation, with load/capacity distribution in accordance with real-life user's supply and demand relations. The simulation results confirm that our proposed solution, called *Decentralized Proactive Resource Allocation* (DPRA) method, can significantly improve the system throughput under the traditional gossip protocol [21, 22] with significantly reduced resource contention. The decision-conflict probability could be limited below 2%, while the system throughput could be improved about 23.6%$\sim$47.1%. Our solution can also keep the system-wide fairness of load distribution up to 97%.

The remainder of this paper is organized as follows. In Section 2, we compare our contribution to related works. We then present our system overview of P2P Grid system and some notations used in our model in Section 3. In Section 4, we analyze the close relationship between system throughput and the new performance metric *average load level*. In Section 5, we propose our resource allocation scheme, including the algorithm skeleton, the customized gossip protocol, the optimal dynamic estimation for the amount of migrated load, and the methodology of avoiding migration conflicts. We report some simulation results in Section 6. We draw conclusion and present future work in Section 7.

2. Related Work

In this section, we discuss related work with respect to three categories: (1) dynamic load balancing algorithms targeting high throughput, (2) gossip protocols adopted in P2P system for dynamic load balancing, (3) node-selection policy that avoids resource contention.

(1) Dynamic load balancing algorithms targeting high throughput

In past decades, throughput usually serves as the major objective when evaluating the Gird's system performance, and load balancing is considered the most important strategy to improve the throughput. For instance, G. Aggarwal, et al. proposed load rebalancing problem [11] and a few approximation algorithms. However, they require central-controlled global task/resource states to make scheduling decisions, which cannot adapt well to dynamic situation, such as the arbitrary node join/departure and various task arrival rates. As such, many fully-decentralized structure based load balancing strategies are also proposed for improving Grid's throughput. Condor-Flock P2P [2] leverages locality-aware heuristics to get short average turnaround time and high system throughput. Another P2P Grid load balancing solution, Rank-based Autonomous Scheduler (RAS) [9], tries to filter the target nodes and tasks with low overhead using a set of rank functions on each node, also aiming to improve system-wide throughput, which was proven to be superior than other meta-heuristics, such as Markov Chain. J. Sonnek et al. [26] propose a reputation-based scheduling approach to improve P2P desktop Grid's system throughput, while maintaining a satisfied success rate of task completion. In [7] and [10], the task scheduling methods also aim at enhancing the system throughput by predicting task arrival rate and node's availability based on history. However, unlike our solution, all of these works are designed completely based on heuristics, which lack accurate asymptotic analyses about the impact of their concentrated performance metrics to the overall throughput, causing the effectiveness of their solutions not easily understood or widely accepted.

(2) Gossip protocols adopted in P2P system for dynamic load balancing

S.K. Kwan, et al. [27] and GDLB [9] both adopt the traditional gossip protocol and also leverage a probabilistic approach for load balancing, which is similar to our work. Our DPRA method adopts a load-status conscious gossip protocol that could more effectively aggregate load states for succeeding load balancing process, significantly outperforming the traditional ones on the improvement of system-wide throughput.

(3) Node-selection policy that avoids resource contention.

Node-selection conflict among the individual schedulers may degrade the load balancing effect severely. O. Sinnen et al. [28] confirmed such a severe resource-contention impact to the task scheduling under their distributed model. The study undertaken in [29] also shows that PlanetLab environment usually experiences the similar problem of “flash crowd” as a growing number of users simultaneously request “slices” on arbitrarily selected nodes and the bursty behavior of users inevitably leads to poor system performance.

Existing load balancing approaches cannot be used in our P2P desktop Grid. A. Bernoit et al. [30] proposed a contention-aware solution with fault-tolerance support, but restricted to a fully-connected network model (such as in in intra-cluster). Such a method is not suitable for large-scale system, in that each peer is not affordable to maintain the global resource states. P. Berenbrink et al. [31] proposed a fully distributed load balancing algorithm, namely *Selfish Load Balancing* based on Game theory. At each scheduling round on each node, every task picks a neighboring host randomly and decides probabilistically whether or not to be migrated towards it. The proposed algorithm can achieve Nash equilibrium with rapid convergence speed without global coordination or cooperation of local schedulers, but it assumes that all hosts are homogeneous. Other negotiation-based method [32] and cooperation-based method [33] can also converge to Nash Equilibrium [34], yet it is subject to the static assumptions which are not realistic in dynamic situations, and cannot guarantee a high-quality system performance. Note that Nash equilibrium does not imply a high system throughput, but only means a stably converged load balancing status. In order to explicitly avoid the resource-contention impact, some solutions [14] resort to remotely checking whether their local task migration decisions would be acceptable by the remote peers. However, such a strategy is undesired since numerous failed confirmations with rejected answers will definitely introduce undesired overhead.

Unlike the existing works, we devise a novel node-selection policy (a.k.a. *stochastic proportional idle resource allocation* (SPIRA)) based on Beroulli trial model, such that the nodes could make their task migration decisions with minimized decision-conflict probability, and this probability can also be derived in theory. Some existing proportional-share scheduling algorithms look similar to our node-selection policy, but they are different in nature. For example, proportional-share policy [35] assumes resources are dividable and users get their shares proportional to their bids, but does not discuss how to minimize the mutual task migration conflict at the task scheduling phase. In contrast, our solution focuses on the problem in the task scheduling phase, such as decision conflict and scheduling delay issue, finally effectively improving the system-wide throughput.

3. Preliminaries

P2P networks can generally be classified into two main categories, unstructured P2P networks and structured P2P networks. Structured P2P networks, such as CAN [36] and Chord [37], are based on Distributed Hash Tables (DHTs). These systems emphasize on a highly structured overlay organization to improve searching performance with low diameter. However, their maintenance overhead is generally too large and the performance would be degraded when peers frequently join/leave the network.

In this paper, we assume all the nodes are organized in an unstructured P2P overlay. A low-cost gossip protocol is adopted for periodically spreading/aggregating resource usage or load status. On each node, the submitted tasks will be placed in a local task queue. We assume that the task scheduling is activated periodically. The local scheduler removes jobs from the local task queue and executes them on the local compute server. The local scheduler also gathers resource state information from its neighbor nodes and decides whether to send tasks to other grid nodes if it is considered overloaded. We assume task arrival rates on nodes are arbitrary, i.e. each node asynchronously receives various number of submitted tasks over time. Thereby, once the scheduler starts, all further arriving jobs are delayed to be scheduled in the next batch.

For simplicity of analysis, we assume there are n heterogeneous nodes in the P2P grid system. Each is denoted as p_i and has a different computing power (or capacity) denoted as c_i, where $1 \leq i \leq n$. We assume the number of tasks received by p_i at the end-time point of each periodic scheduling round is m_i and each task's workload is l_{ik} $(1 \leq k \leq m_i)$. Thus node p_i's total workload l_i is equal to $\sum_{k=1}^{m_i} l_{ik}$. Let $\overline{l_{[t]}}$ refer to the system's average workload at the time point t. The amount of workload l_i will be processed by node p_i at the speed proportional to c_i.

We define $\varphi_i = \frac{l_i}{c_i}$ as node p_i's *load level*. φ_i could also refer to the estimated time cost in processing the remaining workload l_i on p_i. Figure 1 shows how to calculate the load level (φ_i) at each node. At certain time point, the system's *average load level* $\overline{\varphi}$ can be calculated as $\frac{1}{n} \sum_{i=1}^{n} \frac{l_i}{c_i}$.

We define the system throughput as the total workload of completed tasks per

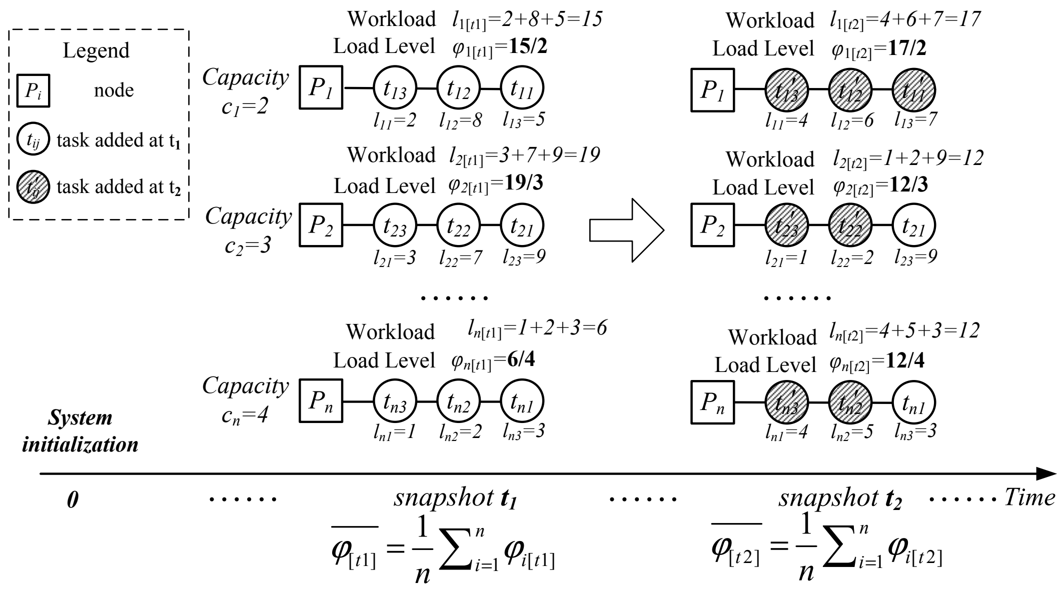

Legend
P_i node
t_{ij} task added at t_1
t'_{ij} task added at t_2
Workload $l_{1[t1]}=2+8+5=15$
Load Level $\varphi_{1[t1]}$=15/2
Capacity $c_1=2$
$l_{11}=2$ $l_{12}=8$ $l_{13}=5$
Workload $l_{2[t1]}=3+7+9=19$
Load Level $\varphi_{2[t1]}$=19/3
Capacity $c_2=3$
$l_{21}=3$ $l_{22}=7$ $l_{23}=9$
Workload $l_{n[t1]}=1+2+3=6$
Load Level $\varphi_{n[t1]}$=6/4
Capacity $c_n=4$
$l_{n1}=1$ $l_{n2}=2$ $l_{n3}=3$
Workload $l_{1[t2]}=4+6+7=17$
Load Level $\varphi_{1[t2]}$=17/2
$l_{11}=4$ $l_{12}=6$ $l_{13}=7$
Workload $l_{2[t2]}=1+2+9=12$
Load Level $\varphi_{2[t2]}$=12/3
$l_{21}=1$ $l_{22}=2$ $l_{23}=9$
Workload $l_{n[t2]}=4+5+3=12$
Load Level $\varphi_{n[t2]}$=12/4
$l_{n1}=4$ $l_{n2}=5$ $l_{n3}=3$
System initialization
0
snapshot t_1
snapshot t_2
Time
$\overline{\varphi_{[t1]}}=\frac{1}{n}\sum_{i=1}^{n}\varphi_{i[t1]}$
$\overline{\varphi_{[t2]}}=\frac{1}{n}\sum_{i=1}^{n}\varphi_{i[t2]}$

Figure 1: Overview/Snapshots of System Workload and Load Level

time unit, i.e. *THRP*(Δt)=$\frac{\sum_{i=1}^{n} l_{i\Delta t}^{(pr)}}{\Delta t}$, where $l_{i\Delta t}^{(pr)}$ refers to the processed workload on p_i in the duration of Δt. In addition, as discussed previously, fairness index of nodes' load levels are commonly considered the key criterion to evaluate the system's load balancing level, and leveraged to design the load balancing algorithms by many other existing researches, but we will theoretically prove that it is insufficient to reflect the system throughput. A good scheduler should not only guarantee all computing resources are used in a balanced and fair manner, i.e., the amount of workload queued on nodes should be proportional to their computing capacities, but also lead to high system throughput eventually.

4. Correlation between System Throughput and Average Load Level

For dynamic P2P Grid systems with pertinent submitted tasks, it is obvious that neither minimum nor maximum load level can reflect the overall throughput in that they only concern about either the lightest or heaviest loaded peer. In this section, we will further prove that the overall system throughput cannot be reflected by Jain's fairness index (Equation (1)) of load distribution either, but fully determined by our proposed *average load level*.

$$F(\varphi) = \frac{(\sum_{i=1}^{n} \varphi_i)^2}{n \cdot \sum_{i=1}^{n} \varphi_i^2} \tag{1}$$

Intuitively, the system throughput is closely related to the average remaining workload ($\bar{l}$) at any time point: smaller remaining workload means more workloads were completed in the past. Yet, it is not trivial to mathematically determine the relationship between *average load level* $\overline{\varphi}$ and system throughput. As follows, we will prove that $\overline{\varphi}$ is always proportional to system's average workload $\bar{l}$ at any time point via the *law of large numbers*.

Theorem 1. *For various arbitrary scheduling algorithms in a large scale P2P grid system (on which each node owns $O(\log(n))$ neighbors without loss of generality*[1]*), given the same initial workload distribution l_1, l_2, $\cdots$, l_n at a specific time point t_1, the algorithm that can achieve smaller* average load level $\overline{\varphi}$, *where $\overline{\varphi} = \frac{\sum_{i=1}^{n} \varphi_i}{n}$, at a later time point $t_2 \Leftrightarrow$ a higher throughput is achieved.*

PROOF. For a single node p_i, let L_i ($1 \leq i \leq n$) denote the random variable of its workload l_i, and Φ_i denote the random variable of its load level φ_i. Then, our proof could be split to three steps: (1) to prove $\frac{1}{n}\sum_{i=1}^{n} L_i \rightarrow \frac{1}{n}\sum_{i=1}^{n} E(L_i)$ and $\frac{1}{n}\sum_{i=1}^{n} \Phi_i \rightarrow \frac{1}{n}\sum_{i=1}^{n} E(\Phi_i)$ for the large scale P2P Grid system. (2) to prove $\frac{1}{n}\sum_{i=1}^{n} E(\Phi_i)$ is always proportional to $\frac{1}{n}\sum_{i=1}^{n} E(L_i)$. (3) to prove Theorem 1: smaller $\overline{\varphi}$ at time point $t_2 \Leftrightarrow$ the algorithm delivers higher throughput.

[1]In general, the number of each node's neighbors (i.e. cache size) cannot increase linearly but with logarithmical scale (i.e. $O(\log(n))$) at most compared to the global system scale (n), otherwise the network traffic maintaining the topology will be intolerable.

Step (1) We will leverage Markov's law of large numbers (LLN) [1] to prove this step. Markov's LLN can be formally described as follows:

If random variables $X_1,\cdots,X_n$ satisfy Condition (2), Equation (3) must hold.

$$\tfrac{1}{n^2}D(\sum_{i=1}^{n} X_i) \to 0, (n \to \infty) \tag{2}$$

$$\lim_{n\to\infty} P(\left|\tfrac{1}{n}\sum_{k=1}^{n} X_k - \tfrac{1}{n}\sum_{k=1}^{n} E(X_k)\right| < \varepsilon) = 1 \tag{3}$$

We will mathematically prove that our large-scale P2P Grid model must satisfy the condition (2), when X_i=L_i or X_i=Φ_i. The means of the proof is based on the fact that any two workloads L_i and L_j (i≠j) are pairwise independent with high probability because the size of each node's local partial-view cache ($\log_2(n)$) is quite tiny compared to the whole system scale (n). We give the formal proof as follows. First of all, we could derive the following equation.

$$\begin{aligned}&\tfrac{1}{n^2}D(\sum_{i=1}^{n} L_i) = \tfrac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} Cov(L_i, L_j) = \tfrac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} \rho_{ij}\sqrt{D(L_i)D(L_j)}\\ &where\ Cov(L_i, L_j)\ means\ the\ covariance\ of\ L_i\ and\ L_j\end{aligned} \tag{4}$$

Note that we aim to compare different algorithms over a specific time period $[t_1, t_2]$, thus there must be an upper bound for L_i and $D(L_i)$, $\forall\, p_i$. In other words, there must exist a constant d such that $\sqrt{D(L_i)D(L_j)} \leq d$ for any pair of nodes p_i and p_j. Hence, we could get following inequality.

$$\tfrac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} \rho_{ij}\sqrt{D(L_i)D(L_j)} \leq \tfrac{d}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} |\rho_{ij}| \tag{5}$$

As for any two single nodes p_i and p_j selected from among all nodes each connected to $O(\log(n))$ neighbors that are organized through unstructured overlay, we assume that each of their neighbors would receive a relatively stable number of

[1]LLN implies that the average of random variables approaches the mean of their mathematical expectations with extremely high probability, if the number of variables is extremely large.

tasks in specific duration (e.g. $[t_1, t_2]$) without loss of generality. Then, the number of tasks migrated from p_i and p_j to outside can be denoted as N_i=$\lambda_i \cdot \log(n)$ and N_j=$\lambda_j \cdot \log(n)$ respectively, where λ_i and λ_j are two constants. Then, $(1-\frac{1}{n})^{N_i}$ is the probability that p_i does not migrate any task to p_j, and $(1-\frac{1}{n})^{N_j}$ is the probability that p_j migrates no tasks to p_i. Hence, the probability that L_i and L_j are mutually non-correlated could be calculated as $(1-\frac{1}{n})^{N_i} \cdot (1-\frac{1}{n})^{N_j}$=$(1-\frac{1}{n})^{N_i+N_j}$. Thereby, we could get Inequality (6).

$$E(\rho_{ij}) \leq (1-\frac{1}{n})^{(N_i+N_j)} \cdot 0 + (1-(1-\frac{1}{n})^{(N_i+N_j)}) \cdot 1 = 1-(1-\frac{1}{n})^{(N_i+N_j)} \quad (6)$$

If we think of ρ_{ij} as random variable, it is obvious that all the n^2 random variables are mutually independent with extremely high probability, thus we could get the following derivation (7) based on *Chebyshev's law of large numbers*, and get Equation (8) based on Inequality (6).

$$\lim_{n\to\infty} P\left(\left|\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}|\rho_{ij}| - \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}E\,|\rho_{ij}|\right| < \varepsilon\right) = 1,\ \forall\varepsilon \quad (7)$$

$$\begin{aligned}
&\lim_{n\to\infty} \frac{d}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}|\rho_{ij}| = \lim_{n\to\infty} \frac{d}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}E\,|\rho_{ij}| \\
&\leq \lim_{n\to\infty} \frac{d}{n^2} \cdot n^2(1-(1-\frac{1}{n})^{(N_i+N_j)}) = \lim_{n\to\infty} d \cdot (1-(1-\frac{1}{n})^{N_i+N_j}) \\
&= \lim_{n\to\infty} d \cdot (1-(1-\frac{1}{n})^{\lambda_i\log(n)+\lambda_j\log(n)}) = \lim_{n\to\infty} d \cdot (1-(1-\frac{1}{n})^{(\lambda_i+\lambda_j)\log(n)})
\end{aligned} \quad (8)$$

We will prove $\lim\limits_{n\to\infty} d \cdot (1-(1-\frac{1}{n})^{(\lambda_i+\lambda_j)\log(n)}) = 0$: In fact, as long as we could prove Equation (9), the above proposition will hold obviously.

$$\lim_{n\to\infty}(1-\frac{1}{n})^{\log n} = 1 \quad (9)$$

Let x = $(1-\frac{1}{n})^{\log n}$, then $\log x$ = $\log n \log(1-\frac{1}{n})$. In following text, we could prove $\lim\limits_{n\to\infty}\log x = 0$ via *L'Hôpital*'s rule in Equation (10), thus $x \to 1$ as $n \to \infty$, which implies that Equation (9) holds.

$$\begin{aligned}
&\lim_{n\to\infty}\log n\log(1-\frac{1}{n}) = \lim_{n\to\infty}\frac{\log(1-\frac{1}{n})}{\frac{1}{\log n}} = \lim_{n\to\infty}\frac{\frac{1}{n^2}\cdot\frac{1}{1-1/n}}{\frac{-1}{\log^2 n}\cdot\frac{1}{n}} \\
&= \lim_{n\to\infty}\frac{-\log^2 n}{n-1} = \lim_{n\to\infty}\frac{-\frac{2}{n}\log n}{1} = -2\lim_{n\to\infty}\frac{1}{n} = 0
\end{aligned} \quad (10)$$

Based on the Formula (4) $\sim$ (10), $\frac{1}{n^2}D(\sum\limits_{i=1}^{n} L_i) \to 0$ as $n \to \infty$, which means our P2P Grid model satisfies the law of large numbers (i.e. Equation (3)). That is, $\frac{\sum_{i=1}^{n} L_i}{n} \to \frac{\sum_{i=1}^{n} E(L_i)}{n}$, as $n \to \infty$. With exactly similar deduction, we could also derive that $\frac{1}{n}\sum_{i=1}^{n} \Phi_i \to \frac{1}{n}\sum_{i=1}^{n} E(\Phi_i)$ as $n \to \infty$.

Step (2) At any time point, $\forall\, i$, we could consider $E(L_i)$ a particular sample, then there will be n sample points (i.e. $E(L_1)$,$E(L_2)$,$\cdots$,$E(L_n)$) in the whole system. We denote the corresponding random variable as L for the n sample points. Similarly, we could also regard Φ as the random variable of n sample points $E(\Phi_1)$, $E(\Phi_2)$, $\cdots$, $E(\Phi_n)$ and think of node capacities c_1, c_2, $\cdots$, c_n as the samples of the random variable C. Note that p_i's capacity may be different from p_j's, yet any particular node's capacity is always fixed (i.e. c_i will not be changed over time). Thus, we could get Equation (11), according to the numerical characteristics of random variables, which implies $E(\Phi) \propto E(L)$ at any time point.

$$E(\Phi) = E(L \cdot \tfrac{1}{C}) = E(L) \cdot E(\tfrac{1}{C}) \qquad (11)$$

In addition, obviously, $E(L) \to \frac{1}{n}\sum_{i=1}^{n} E(L_i)$ and $E(\Phi) \to \frac{1}{n}\sum_{i=1}^{n} E(\Phi_i)$, as n is extremely large (e.g. $n \to \infty$). Hence, $\frac{1}{n}\sum_{i=1}^{n} E(\Phi_i)$ is proportional to $\frac{1}{n}\sum_{i=1}^{n} E(L_i)$ based on Equation (11), when n is large.

Step (3) With the results of Step (1) and Step (2), we could easily get $\overline{\varphi} \propto \overline{l}$ at any time point (such as t_2). Since fewer remaining total amount of workload (i.e. smaller $\overline{l}$) implies that more workloads are already processed, smaller $\overline{\varphi} \Rightarrow$ higher system throughput. The above deduction's direction can also be inverse, thus smaller $\overline{\varphi} \Leftrightarrow$ higher throughput, i.e., smaller $\overline{\varphi}$ is a necessary and sufficient condition for achieving higher system-wide throughput. $\square$

Note that the Markov's LLN does not require the random variables (either L_i or Φ_i for any i) are independent or conform to some identical stochastic process or distribution, thus the above proposition is of high practicability. According to the Formula (1), the fairness index $F(\varphi)$ can be rewritten as $\frac{n\overline{\varphi}^2}{\sum_{i=1}^{n}\varphi_i^2}$, that is, $\overline{\varphi}$=$\sqrt{F(\varphi)\cdot\frac{\sum_{i=1}^{n}\varphi_i^2}{n}}$. Obviously, the fairness index $F(\varphi)$ cannot fully determine $\overline{\varphi}$ because $\overline{\varphi}$ is also related to $\frac{\sum_{i=1}^{n}\varphi_i^2}{n}$, though it could be used to evaluate the load balancing level [14, 18, 20]. Thus, the system throughput cannot be sufficiently determined by the fairness index of load level, but the average load level.

In order to simplify our design, we convert such an average load level criterion to a more tractable criterion σ_Φ (=$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\varphi_i-\overline{\varphi})^2}$), i.e. standard deviation of load level, whose variance can be proved closely related to the average load level (also to system throughput), as follows.

Corollary 1. *For different arbitrary load balancing algorithms leading to the same average load level when running them on the uniform P2P Grid system*[1] *in turn, Jain's fairness index of load level increases iff σ_Φ decreases.*

PROOF. Fairness index was derived from the overall standard deviation (i.e. $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\varphi_i-E(\Phi))^2}$) as proposed in [18]. According to the proof in Step (2) of Theorem 1, $E(\Phi)$ in a large scale P2P Grid is close to $\overline{\varphi}$. As such, $\sigma_\Phi = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\varphi_i-\overline{\varphi})^2} = \sqrt{\frac{1}{n}(\sum_{i=1}^{n}\varphi_i^2-n\overline{\varphi}^2)}$, when n is large. Let A=$n\overline{\varphi}^2$ and B=$\sum_{i=1}^{n}\varphi_i^2$, then σ_Φ=$\sqrt{\frac{1}{n}(B-A)}$. On the other hand, $F(\varphi)$=$\frac{n\overline{\varphi}^2}{\sum_{i=1}^{n}\varphi_i^2}$=$\frac{A}{B}$. Since different algorithms here are assumed to achieve the same average load level (i.e. A is fixed), it is obvious that smaller σ_Φ will lead to smaller B, thus getting higher fairness index $F(\varphi)$, and vice versa. □

Corollary 2. *For different arbitrary algorithms leading to the same fairness index of load level when running them on the same P2P Grid system in turn, the average*

[1]the "uniform" system indicates that each nodes' initial remaining workloads and its following task submission events are exactly the same in different experiments with various algorithms.

load level decreases iff σ_Φ decreases.

PROOF. Here, we omit the proof because it is similar to that of *Corollary* 1. □

According to the above theorems and corollaries, the smaller standard deviation of load level (σ_Φ) is, the higher fairness index or system throughput we can get. Hence, we design our scheduling algorithm by focusing on σ_Φ for simplicity. The details will be described later.

5. Decentralized Task Scheduling in P2P Grid

This section presents the framework of our decentralized task scheduling scheme, including a load-status conscious gossip protocol and the basic design of our decentralized task scheduling algorithm for large-scale P2P Grid systems.

5.1. Newscast-based Gossip Protocol

Rather than flooding protocols, we adopt a low-cost gossip protocol [21] to periodically disseminate nodes' states for fast resource discovery. To minimize the transmission overhead, each peer node will disseminate a message containing its expected amount of workload to be shifted onto this node (i.e. $M(p_i)$ to be introduced later) and its IP address. Each node will also forward the gossip messages received from other nodes to a few randomly selected neighbors per gossip cycle. In our design, we limit the number of neighbors (denoted as δ) to be $\log(n)$ and select the neighbors according to the *Newscast* peer sampling model [38] (to be described later). By continuously exchanging messages, each node will accumulate a set of resource messages (i.e. a set of nodes' state records) whose availabilities are determined by a Time-To-Live (TTL) value. All the state records stored by one node form this node's *acquaintance node set*, which varies over time. Note that a node's acquaintance nodes may not be its neighbor, but can still be connected through network. The number of acquaintance nodes on average at each node could be controlled by the TTL of the state messages.

Newscast-based gossip protocol [38] has been widely used because of its exponential convenience speed on information aggregation. Newscast [38] is a kind of neighbor-updating policy, under which the new neighbors of each node will be randomly selected from the merging set composed of the current neighbors and the neighbors of one of its neighbors. M. Jelasity et al. [38, 39] proved that periodically performing a few computations (such as calculating mean value, max/min, top-*K* filtering, standard deviation, etc.) at each node on the computational results calculated by the Newscast-based neighbors could exponentially aggregate the global-area statistical results with little error.

Unlike such a Newscast-based gossip protocol [38], we designed a *load-status conscious* gossip protocol which could deliver even more effective message propagation efficiency in P2P desktop Grid. To ensure fast delivery of the state messages to the right target nodes and wide spreading of the information, the message whose source node is heavily loaded will be forwarded to $\delta/2$ most lightly loaded neighbors selected from the *acquaintance node set* and the other $\delta/2$ randomly selected acquaintance nodes based on Newscast policy. Similarly, if the message's source node is marked as *lightly loaded*, the message will be delivered to $\delta/2$ heavily loaded acquaintance nodes and $\delta/2$ random acquaintance nodes at each gossip cycle. Hence, each node p_i should periodically classify its *acquaintance node* set into *lightly loaded node set* (*UL*(p_i)) and *heavily loaded node set* (*OL*(p_i)), by calculating their average workload level. TTL is set by the source node to indicate the duration of validity of node state recorded in the message. It will be dropped immediately upon the expiration of specified TTL, or when a more updated message from the same source node is received. With the state information accumulated through the changing neighbor nodes, each heavily loaded node will periodically relocate its unaccomplished tasks to other nodes with less workload to mitigate its over-utilized status. In the following text, the interval between two such consecutive load rebalancing steps is called *epoch* (i.e. the time-slot between two successive scheduling rounds), which is much longer than a gossip cycle.

5.2. Decentralized Proactive Resource Allocation

Based on the proposed gossip framework, our decentralized task scheduling algorithm performs as follows. Algorithm 1 below shows the key steps performed on each node p_i at the end of each epoch: (1) Every node p_i computes an appropriate load amount $M(p_i)$ to be relocated (Line 2) according to p_i's task arrival rate (Details are discussed in Section 5.3). (2) Once the current peer p_i is detected heavily loaded (Line 3), i.e. $M(p_i)$ is greater than a threshold ε, the algorithm will selectively migrate outward a set of surplus tasks, denoted as *T_Set* (Line 5). The node selection policy determines the location of the migrated tasks and the selection probability is computed based on a *conflict-minimizing strategy*, i.e., stochastic proportional idle resource allocation (SPIRA) algorithm to be described in Section 5.4. The main objective of the SPIRA algorithm is to avoid the decision conflict among the autonomic load migrations initiated independently by peer nodes. Note that in our design, task migrations are only initiated by the heavily loaded nodes, in that push-based mechanism is easy to conduct and can quickly adjust to the state changes of hot spots [40].

The *UL*(p_i) at Line 4 refers to a set of underloaded nodes known to node p_i. Likewise, *OL*(p_i) stands for the set of overloaded nodes known to p_i. The members of the two sets could be changed at the end of each epoch after gathering more information from its neighboring nodes. Via conflict-minimizing method (Line 7∼8), the algorithm will migrate a number of tasks with an approximate amount of $M(p_i)$ total load to other nodes. A utility function *Pol*(S_f, R) is used for the task selection, where S_f refers to the *utility function set* and R is the corresponding rule that sets the constraints. Both S_f and R can be customized by users/designers based on their application demands. A typical example is designing a threshold to filter the tasks with high migration cost.

Algorithm 1 SKELETON OF DPRA ALGORITHM

Notice: This algorithm is executed on each peer p_i (0≤i≤n).

Input: lightly loaded node set (*UL*(p_i)) and heavily loaded node set (*OL*(p_i))

Output: migration decisions

```
 1: while (TRUE) do
 2:   Calculate the load amount $M(p_i)$ to be moved based on $p_i$'s task arrival rate.
 3:   if ($M(p_i)>\varepsilon$) then
 4:     Compute selection probability distribution $\delta$ for every peer in UL($p_i$) based on conflict-
        minimizing strategy (SPIRA algorithm described in Section 5.4).
 5:     Compose task set (T_Set) filtered by Pol($S_f$, $R$), subject to $\sum_{l_{ik} \in T\_Set} l_{ik} \approx M(p_i)$.
 6:     for (each task $\in$T_Set) do
 7:       Select one target lightly loaded node $p_j \in UL(p_i)$ according to its selection probabil-
          ity $P(p_j)$ calculated at Line 4.
 8:       Migrate the task to $p_j$.
 9:     end for
10:   end if
11:   Wait until next round (i.e. epoch).
12: end while
```

5.3. Estimation of $M(p_i)$

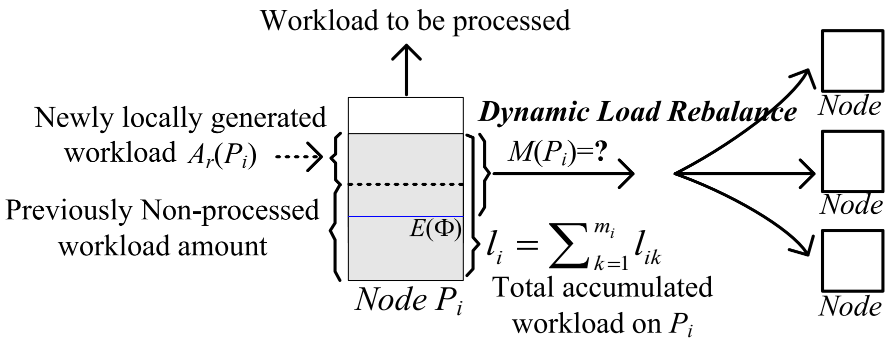
Let $A_r(p_i)$ denote the arrival rate of workload, i.e. the amount of load to be submitted/generated onto the node p_i by its user. $P_r(p_i)$ refers to the processing rate (flops/s), i.e. the amount of load processed by p_i in each epoch. To avoid overloading and keep the system stable, the amount of load beyond its processing capability $P_r(p_i)$, i.e., $M_r(p_i)$=$A_r(p_i)$−$P_r(p_i)$, should be shifted duly from node p_i to other nodes [41]. Therefore, the expected load status of p_i at the jth epoch could be formulated as Equation (12), where $A_r^{(j)}(p_i)$ and $P_r^{(j)}(p_i)$ are the amounts of workload submitted and processed on the node in the jth epoch, respectively.

$$M_r^{(j)}(p_i) = A_r^{(j)}(p_i) - P_r^{(j)}(p_i) \tag{12}$$

If the above equation could always be maintained on each node, the global system will be always kept in a load balancing status stably. However, $M_r^{(j)}(p_i)$ is likely not obtainable, due to many unpredictable factors, such as the inevitably unstable network status, abruptly disconnected remote nodes, or their access control restrictions. Hence, we also leverage the snapshot based load rebalancing strategy to further tune the migration errors. In general, nodes' states are fuzzily classified to two levels: *overloaded* and *underloaded*, which could be varied over time. They are defined in Formula (13), where $\overline{\overline{\varphi}}$ is approximated as $(\sum_{p_j \in as(p_i)} l_j)/(\sum_{p_j \in as(p_i)} c_j)$ and $as(p_i)$ means p_i's acquaintance node set. As $\overline{\overline{\varphi}}$ is computed based on the load information gathered from the acquaintance node set, it only reflects a local view on the global load balancing status.

$$\begin{cases} p_i \ is \ underloaded \cdots\cdots \varphi_i < \overline{\overline{\varphi}} \\ p_i \ is \ overloaded \cdots\cdots\cdot \varphi_i > \overline{\overline{\varphi}} \end{cases} \tag{13}$$

The basic idea of our *dynamic load rebalancing* scheme is that, once a peer p_i is detected *overloaded*, it should shift an amount of surplus workload by considering the workload amount to be generated locally in a short term. Figure 2 shows the workload on node p_i (including the accumulated non-scheduled tasks and newly generated workload) at jth scheduling round.



Workload to be processed
Newly locally generated workload $A_r(P_i)$
Previously Non-processed workload amount
$E(\Phi)$
Node P_i
Dynamic Load Rebalance
$M(P_i)$=?
$l_i = \sum_{k=1}^{m_i} l_{ik}$
Total accumulated workload on P_i
Node
Node
Node

Figure 2: Load Migration on a Peer Node

We let $P_{rr}^{(j)}(p_i)$ denote the real load amount processed by p_i at jth epoch and $M_{rr}^{(j)}(p_i)$ denote the real load amount moved out from p_i at jth epoch. Equation (14) and (15) present the load amount (denoted by $M_s^{(j)}(p_i)$) to be migrated from p_i at jth epoch with respect to the load rebalancing policy, where $A_r^{(j)}(p_i) - P_{rr}^{(j)}(p_i) - M_{rr}^{(j)}(p_i)$ refers to the remaining load amount at the end of jth epoch and $\theta \cdot c_i^2$ is an extra part aiming to reach the minimal deviation of load level and $\theta = (\sum_{p_j \in as(p_i)} l_j - E(\Phi) \cdot \sum_{p_j \in as(p_i)} c_j) \,/\, \sum_{p_j \in as(p_i)} c_j^2$, which will be proved in Appendix via convex optimization.

$$\varphi_i^{(j)} = \begin{cases} \gamma & \gamma > 0 \\ 0 & \gamma \leq 0 \end{cases} ,where\ \gamma = \frac{A_r^{(j)}(p_i) - P_{rr}^{(j)}(p_i) - M_{rr}^{(j)}(p_i)}{c_i} + \varphi_i^{(j-1)} \tag{14}$$

$$M_s^{(j)}(p_i) = (\varphi_i^{(j)} - E^{(j)}(\Phi)) \cdot c_i - \theta \cdot c_i^2 \tag{15}$$

By taking into account the estimated task arrival rate and processing rate of next epoch (i.e. Equation (12) at (j+1)th epoch), the final amount of load that needs to be shifted from node p_i at jth epoch is given in Formula (16).

$$M^{(j)}(p_i) = M_r^{(j+1)}(p_i) + M_s^{(j)}(p_i) \tag{16}$$

When $M^{(j)}(p_i) > \varepsilon$, node p_i needs to move relatively surplus loads outward, a.k.a. *proactive overloaded* status. On the contrary, when $M^{(j)}(p_i) < -\varepsilon$, it is at*proactive underloaded* status, thus p_i is expected to receive more loads from outside. Moreover, if $|M^{(j)}(p_i)|\leq\varepsilon$, this indicates that p_i's state is *proactive balanced.* ε set by node p_i is equal to the smallest task load on p_i. In the following text, node's status (overloaded, underloaded, or balanced) always means under the proactive manner, thus we will omit the term "proactive" for short. Compared to the traditional load rebalancing strategy [11] that only reschedules the load based on $M_s^{(j)}(p_i)$ on node p_i, our decentralized proactive allocation scheme also considers $M_r^{(j+1)}(p_i)$ at the jth epoch, which can achieve better adaptability.

An additional question is how to determine $A_r(p_i)$ for the next epoch. In reality, its value could be predicted based on the historical records about users' task submission rates within specific periods [42, 43, 44] or tasks' execution patterns [45, 46, 47], and the prediction errors could already be limited down to 10% against the real workloads [47]. Note that the prediction methods are beyond the scope of this paper, and our main effort is combining the above proactive design with our conflict-minimizing strategy (to be discussed in Section 5.4), such that the system throughput will not be degraded notably even with 50% margin of error on estimating $A_r(p_i)$, as validated by our simulation.

5.4. Conflict Minimization Method

One challenge in the decentralized resource allocation is the decision conflict problem: the unilateral task migration decisions may result in a lightly loaded node suddenly becomes heavily loaded (a.k.a. *conflict decision event*) because multiple overloaded nodes prone to select the same underloaded peer to process their surplus loads due to the lack of coordination.

To minimize the impact on conflict decision, each heavily loaded peer assumes its underloaded peers (i.e. *UL*(p_i)) and overloaded peers (i.e. *OL*(p_i)) are also known by other heavily loaded peers. Every peer will estimate the probability of migration conflict based on its local view and make the best-response decisions to avoid conflicts.

The proposed conflict-minimizing method, namely Stochastic Proportional Idle Resource Allocation (SPIRA), consists of three key steps:
On every overloaded peer p_i:
1) Compute *SUL(*p_i*)*=$\sum_{p_k \in UL(p_i)} |M(p_k)|$
2) For each node p_k in $UL(p_i)$, compute its selection probability $P(p_k) = \frac{|M(p_k)|}{SUL(p_i)}$.
3) Move selective surplus tasks to node p_k stochastically according to $P(p_k)$.

Without loss of generality, we could view the process of selecting target underloaded nodes by overloaded nodes in a fully-distributed competitive P2P Grid environment as a number of independent Bernoulli trials[1]. That is, the distributed node selection process for load balancing can be analyzed equally as follows:
(1) Divide all peers with un-balanced load into overloaded nodes and underloaded nodes, constructing a bipartite graph.
(2) Each overloaded peer identifies its surplus tasks and reassign them to the underloaded peers based on a uniform selection probability (by SPIRA method).
(3) Evaluate if each assignment is a viable task migration event (i.e, a success in Bernoulli trial) or *conflict decision* event (i.e., a failure).

Figure 3 illustrates the idea in a simplified scenario. All the surplus tasks will be migrated to lightly loaded nodes stochastically according to a uniform probability distribution, $\{\frac{30}{100}, \frac{20}{100}, \frac{30}{100}, \frac{20}{100}\}$.

To reduce the communication cost, each gossip message just contains the value

[1]In the probability theory, Bernoulli trial is defined as an experiment whose random outcome is alternative, either “success” or “failure”.

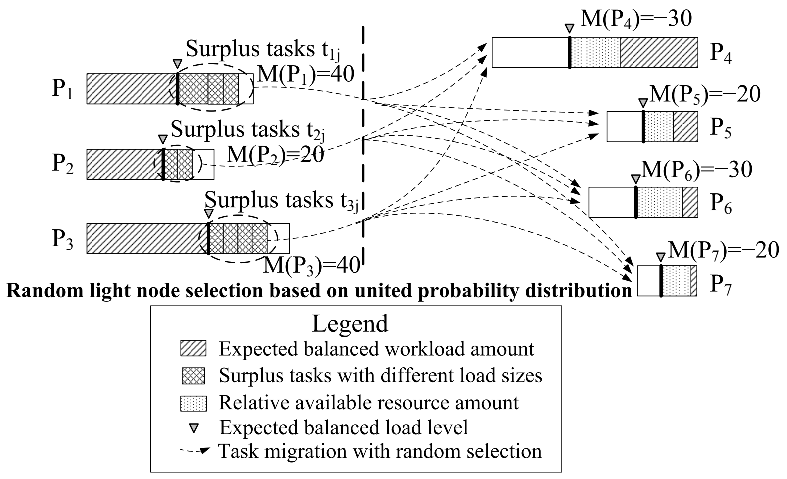

M(P4)=−30
Surplus tasks t1j
M(P1)=40
P1
P4
M(P5)=−20
Surplus tasks t2j
M(P2)=20
P2
P5
M(P6)=−30
Surplus tasks t3j
P6
P3
M(P3)=40
M(P7)=−20
Random light node selection based on united probability distribution
P7
Legend
Expected balanced workload amount
Surplus tasks with different load sizes
Relative available resource amount
Expected balanced load level
Task migration with random selection

Figure 3: Decentralized load balancing can be viewed as a set of Bernoulli trials

of $M(p_i)$ and the source node's identifier/IP. Note that, if each message's TTL (i.e. the number of gossip cycles it lasts) is equivalent to the duration of one epoch, the load state cached on other nodes in the jth epoch should always be equal to $M^{(j)}(p_i)$, rather than $M^{(j-1)}(p_i)$ or $M^{(j+1)}(p_i)$. This guarantees that the state information for task (re)scheduling is always up-to-date.

The time complexity of SPIRA method is $O(|UL(p_i)| + m_i)$ compared to $O(|UL(p_i)|\cdot log(|UL(p_i)|)+m_i\cdot\log(m_i))$ of other sorting based solutions [14, 11], because the overall scheduling effect has nothing to do with the sorting order of tasks to be assigned on each node. As follows, we carefully analyze the decision-conflict probability delivered by the SPIRA strategy.

Theorem 2. *Given that every heavily loaded node moves its surplus load to underloaded nodes selected by the SPIRA method from its acquainted nodes, the probability of the decision conflict event on lightly loaded node p_k, denoted as $P_{DC}(p_k)$, should be no greater than $\frac{(xP(p_k))^{L_k+1}}{(L_k+1)!}$ (Formula (17)).*

$$P_{DC}(p_k) \approx 1 - e^{-xP(p_k)} \sum_{i=0}^{L_k} \frac{1}{i!}(xP(p_k))^i \prec \frac{(x \cdot P(p_k))^{L_k+1}}{(L_k+1)!} \qquad (17)$$

where L_k=$s \cdot P(p_k)$, x is the total number of tasks to be shifted from all heavily loaded nodes and $\prec$ stands for "approaches or smaller than".

PROOF. The proof is omitted here and please reference our previous publication [48] for the details. Since our model in this paper adopts the proactive idea on load migration (i.e. $M(p_k)$ to be migrated from p_k) rather than the static surplus load amount $(ilf(p_k) \cdot c_k)$ used by [48] without proactive support, the proof will also be revised a little bit, but the conclusion will not be changed at all.

Based on the Formula (17), different x will get different probability on decision conflict event, while the value of x is dependent upon the amount of load every heavily loaded node dispels. If each heavily loaded node p_h dispels $\theta \cdot M(p_h)$ (where $0<\theta\leq 1$), then $x = \theta \cdot s$. As an example, when every heavily loaded node migrates only $\frac{1}{3}$ of surplus load (i.e., θ=$\frac{1}{3}$) and node p_k's L_k=10, $P_{DC}(p_k) \prec \frac{(10/3)^{11}}{11!}$ $\approx 1.41\%$.

We also estimate the overall conflict probability by calculating the system-wide conflict events (denoted by $E_{DC}(x)$) over the total number of tasks to be migrated from all heavily loaded nodes (i.e., x(=θs)), as shown in Formula (18).

$$E_{DC}(x) = x \sum_{p_k \in UL} P(p_k) \cdot P_{DC}(p_k) \tag{18}$$

The overall conflict probability is very small according to Equation (18). Suppose there are 140 heterogeneous lightly loaded nodes (p_1,p_2,$\cdots$,p_{140}) whose relatively idle resource amounts $M(p_i)$ (calculated compared to $E(\Phi)$) are -1, -2, $\cdots$, -140, where the total number of tasks to be migrated from heavily loaded nodes is $\theta \cdot s$=$\theta \cdot \sum_{i=1}^{140} i$=$\theta$·9870. Then, Table 1 presents the estimated probability of conflict events based on Equation (18), and the conflict probability could be down to 0.025 if each heavily loaded node migrates 80% surplus load outward. We will also evaluate the performance via simulation in Section 6.

Table 1: Conflict event Probability when migrating 9870 tasks to 140 nodes

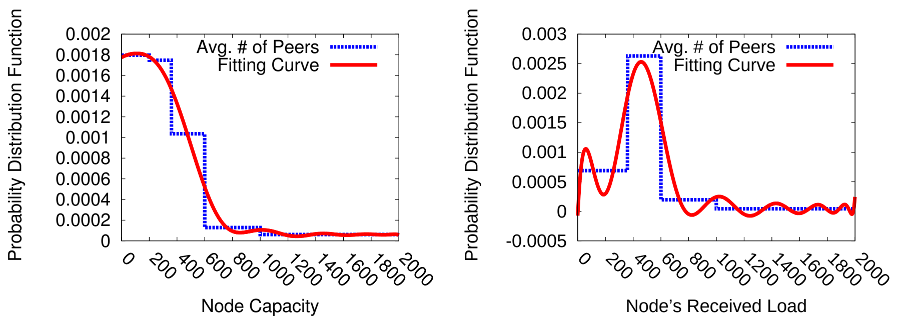
θ	**probability**	θ	**probability**	θ	**probability**
1	0.470	0.95	0.288	0.9	0.148
0.85	0.064	0.8	0.025	0.75	0.010
0.7	0.004	0.65	0.001	0.6	9.545×10^{-4}
0.55	4.918×10^{-4}	0.5	2.617×10^{-4}	0.45	1.419×10^{-4}
0.4	7.755×10^{-5}	0.35	4.216×10^{-5}	0.3	2.250×10^{-5}
0.25	1.157×10^{-5}	0.2	5.573×10^{-6}	0.15	2.393×10^{-6}

6. Performance Evaluation

6.1. Experimental Setting

For each experiment, we first construct an emulated physical network with n ($1000 \sim 8000$) computers randomly connected with various bandwidths by *Waxman* model through Brite topology generator [25]. Over this construction, we then use PeerSim [49] to simulate the asynchronous events based on the workload/capacity distribution reported by PPlive system [50]. The reason we choose PPlive's statistics is that they truthfully represent the heterogeneous supply and demand relationship among daily-life P2P users, which we deem is important to our experiment. Note that one node's capacity here is referred to as its buffering capability, which is set proportional to its processing ability. Through the 25-degree polynomial fitting function, we compute the de facto distribution in Figure 4.

According to M/M/1 model, the rate of loads added to every peer is various in terms of *Poisson* process with a varied λ determined by workload distribution. We make the load of each task equal to 2,4,8,16 or 32 million instructions (MI). For clear observation, $\sum_{i=1}^{n} A_r(P_i)$ is kept equal to $\sum_{i=1}^{n} P_r(P_i)$ in each snapshot and c_i is set equal to processing rate ($P_r(P_i)$), whose mathematical expectation is



(a) Capacity Distribution (b) Workload Distribution

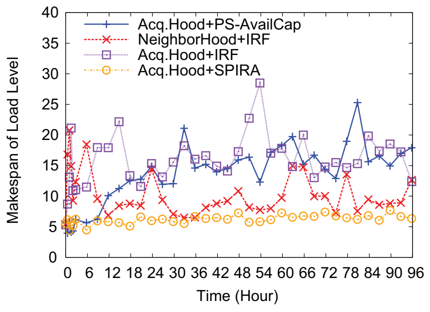
Figure 4: Capacity and Workload Distribution

10 MIPS. Each peer's neighbor degree is $\log(n)$.

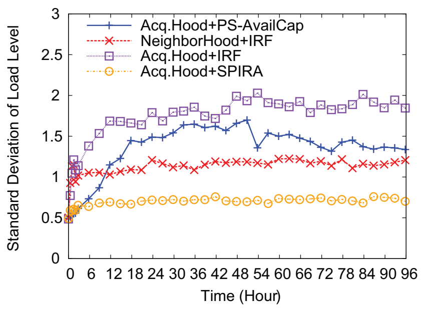
During every rescheduling interval (i.e. epoch), each peer will be injected by new tasks whose total amount is equal to $A_r(p_i)$ and some workloads on it will also be processed (i.e. $P_{rr}(p_i)$). The value of the real processed portion $P_{rr}(p_i)$ is set based on Poisson distribution whose λ=$P_r(p_i)$. Each peer node performs either DPRA scheme or other comparative algorithms periodically in every epoch and each epoch contains 6 gossip cycles. Each gossip cycle is set to 20 seconds.

6.2. Experimental Result

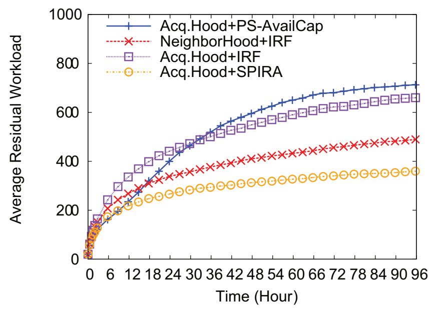
Figure 5 shows the makespan, standard deviation, fairness index of load level and average residual workload (i.e. $\bar{l}$) in the duration of 4 days, where the system scale is 1000 peers and the resource discovery method uses traditional gossip protocol (the following experiments will use the same setting unless special statement). The idlest resource first (IRF) policy (e.g. min-min and max-min [51, 52]) with complete global information have been widely adopted by many distributed resource allocations [51]. Specifically, the decentralized min-min (max-min) algorithm makes every node independently search its surrounding lightest loaded peer and move the minimum (maximum) task to it. PS-AvailCap [27] is similar to our approach, but its node selection probability is simply set proportional to the idle host p_k's availability (i.e. $c_k - l_k$) instead of the relative idle level (i.e. $c_k \cdot \varphi_k - l_k$) adopted by our load rebalancing approach (i.e. $M_s(p_k)$) (referred to as *SPIRA* in the Figure 5).



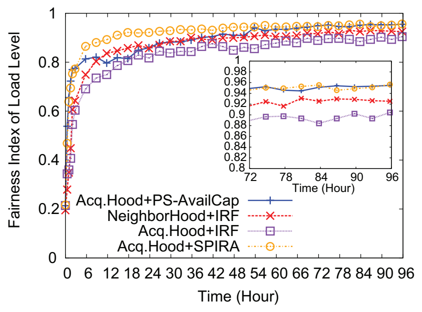
(a) Makespan



(b) Standard Deviation



(c) Average Residual Workload



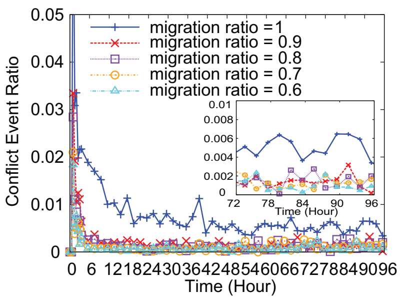
(d) Fairness Index

Figure 5: Allocation of Different Algorithms

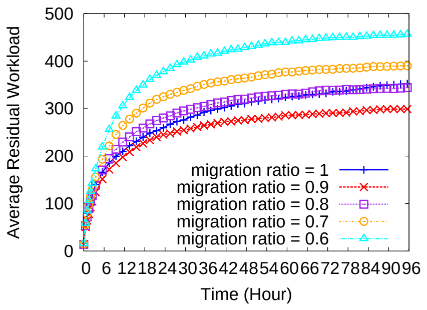
In Figure 5, *NeighborHood* curve indicates that each heavily loaded node just selects the candidate lightly loaded peers from its direct neighbor-peer set, while *Acq.Hood* means that each heavily loaded node's selection range is the larger set with multi-hop based lightly loaded acquaintance peers. According to the analysis in Section 4, higher fairness index of load level or lower standard deviation implies higher balanced degree of workload status; lower makespan of load level means higher stability; lower overall average residual workload (i.e. $\bar{l}$) explicitly indicates the higher throughput. In our simulation, we observe that the fairness index without any load balancing strategy will be kept only about 0.2, thus all the four solutions deliver acceptable fairness index based on Figure 5 (d). Nevertheless, both idlest resource first (IRF) policy and PS-Availability policy present poor allocation results, e.g. relatively high makespan, standard deviation, and average residual workload. This is mainly due to the decision conflict problem amongst the competing peers. In contrast, our conflict-minimizing strategy shows much better result due to its mitigated decision conflicts (about 23.6%$\sim$47.1% reduction on residual workload in terms of Figure 5 (c)).

As proved in Theorem 2, the lower migration ratio is, the lower decision conflict probability is. However, lower migration ratio may not lead to higher system throughput or load balancing level, because the lower migration ratio will definitely make the heavily loaded nodes not reach the expected average load level ($\overline{\varphi}$), either do the lightly loaded nodes. As a result, there must be a tradeoff between the migrated load amount and the decision conflict. Figure 6 shows the load balancing result under different load migration ratios (i.e. θ). Figure 6 (a) shows that the decision conflicts are mitigated with decreasing value of θ. The system throughput (i.e. average residual workload) gets the optimal result when θ approaches 0.9.

Figure 7 presents the resource allocation result under different resource discovery protocols during 4-days simulation (also based on optimized migration load amount calculated by Equation (16)). Figure 7 (a) shows that our load-status conscious gossip protocol can significantly improve the system throughput, while the fairness index will not change notably, as shown in Figure 7 (b). This also confirms our conclusion that fairness index cannot determine system throughput.

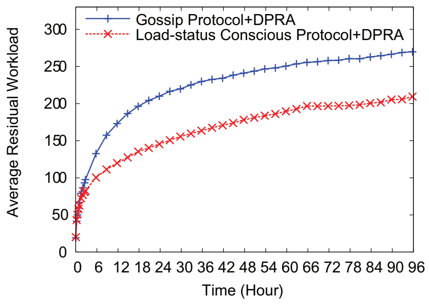


(a) Conflict Event Ratio

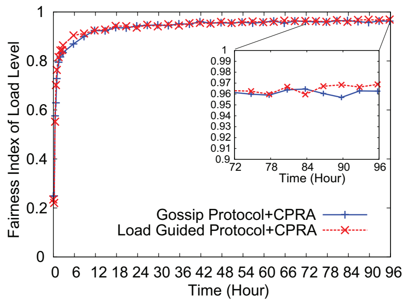


(b) Average Residual Workload

Figure 6: Allocation under Different Load Migration Ratios



(a) Average Residual Workload

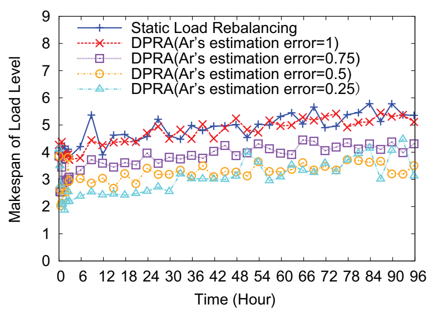


(b) Fairness Index

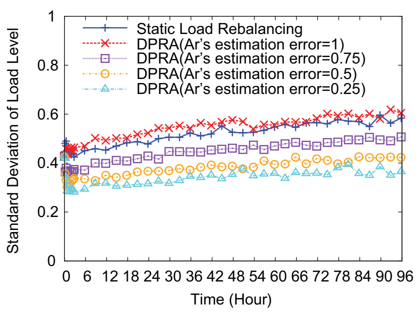
Figure 7: Allocation with Different Message Propagation Protocols

All the rest tests are carried out via the load-status conscious gossip protocol.

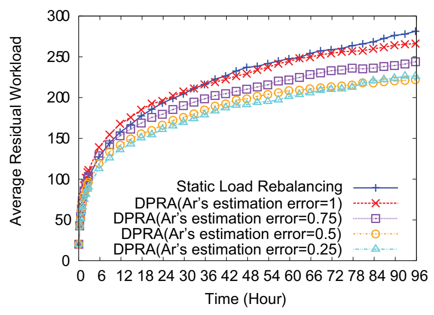
Figure 8 shows the results with different errors on the proactive estimate of $A_r(p_i)$, and this error is defined as $(1 - \frac{estimated\_task\_arrival\_rate}{the\_exact\_task\_arrival\_rate})$. Compared to the traditional load rebalancing framework, DPRA shows significantly improved performance by considering newly arriving tasks. Moreover, the under-estimation with $25\%{\sim}50\%$ ratio shows no performance degradation than with exact task arrival rate estimation. This is because every peer adopts the Bernoulli trials with the selection-probability proportional to its viewed idle resource amounts based on our conflict-minimizing strategy. That is, in accordance with the Theorem 3, the less surplus load amount migrated, the less conflict probability is. This could



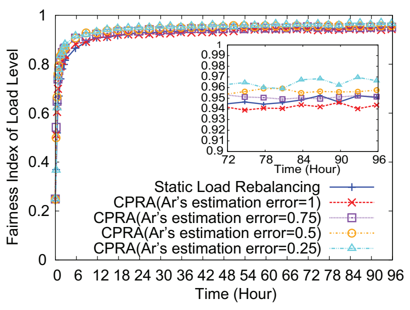
(a) Makespan



(b) Standard Deviation



(c) Average Residual Workload



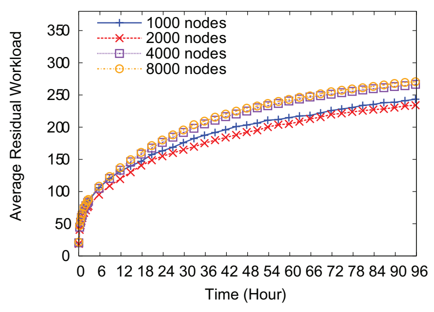
(d) Fairness Index

Figure 8: Allocation with Different Proactive Levels

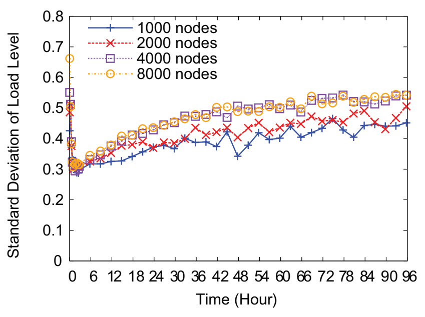
make up the negative effect of the estimation error in return to a certain extent. Henceforth, $A_r(p_i)$ would better be based on conservative estimate of arrival rates in various time periods.

Figure 9 presents the DPRA's result with different scales. With increasing number of peers, it will still converge exponentially with slight errors. This is because that our conflict-minimizing method adopts a probabilistic sharing mechanism (i.e. *Theorem* 2) under which every node just needs to cache a few amount of information.

We also evaluate our DPRA algorithm in unstable (i.e. peer-churning) environment with a percentile of joining/leaving nodes. *0.X dynamicity* means X percent of peers are replaced by new ones every gossip cycle, i.e. $1-(1-0.X)^3$ nodes will be changed in every allocation interval. For instance, "0.2 dynamicity"



(a) Average Residual Workload

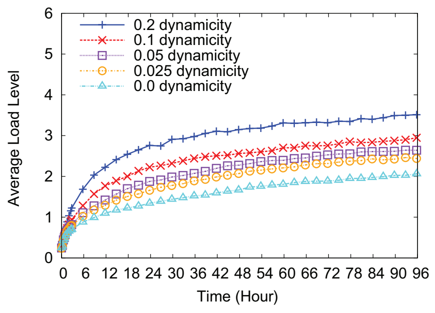


(b) Standard Deviation

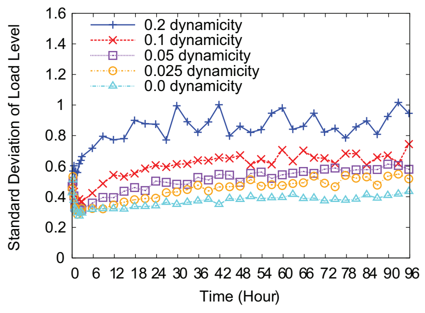
Figure 9: Scalability of DPRA

means about half of nodes have to be replaced between two reallocations. From the results in Figure 10, we can see that DPRA can adapt to node-churning well. Most importantly, from Figure 10 (a) & (c), we observe that the average load level is indeed consistent with average residual workload, which confirms our *Theorem* 1 proved in Section 4.

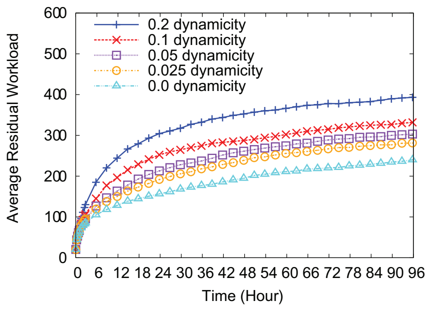
Finally, we compare the performance of the DPRA algorithm with and without migration overhead. Just as shown in Algorithm 1, we adopt the filtering policy to filter out tasks with too heavy migration overhead. The task sizes are 2,4,8,16,32 or 64 MI, and the task migration overheads are set to two minutes on average, thus the communication to computation ratio can be calculated as about $2{:}\frac{126}{6\times 10}\approx\frac{1}{1}$. In our simulation, when any node makes any task migration decisions, it will also notify the corresponding destination node to change its workload, so that it could notify its neighbors about its updated state as soon as possible in a proactive manner. From the Figure 11, we can clearly observe that neither the average remaining workload nor fairness index of the load level will be changed notably.



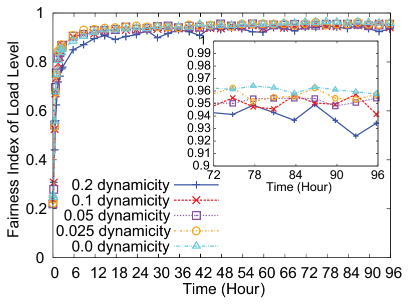
(a) Average Load Level



(b) Standard Deviation



(c) Average Residual Workload

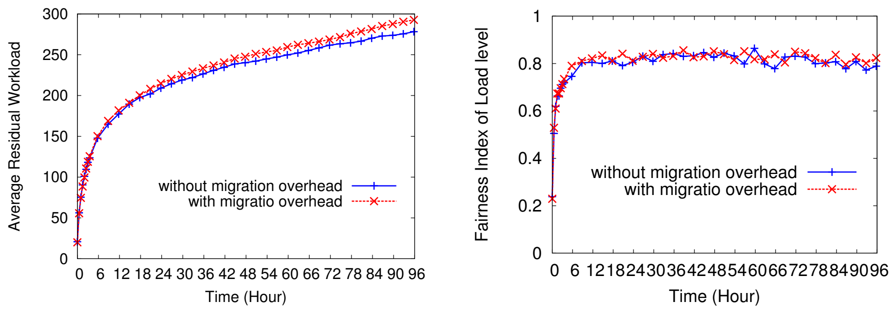


(d) Fairness Index

Figure 10: DPRA in Unstable Environment

7. Conclusion and Future Work

To our knowledge, this is the first attempt to prove that fairness index of load level is insufficient to determine the system throughput of large-scale P2P Grid. Based on this theory, we design a novel resource allocation scheme (DPRA) by combining the dynamic estimation of arrival tasks and the load rebalancing policy, in order to maximize the system throughput of P2P Grid. We also prove that the ratio of migration load amount from every heavily loaded node may significantly impact the mutual decision conflict. Via simulation, we confirm that our method with conflict-minimizing strategy and the load-status conscious gossip protocol can not only effectively balance the system-wide workload among heterogeneous peers with satisfactory converged fairness index (up to 97%), but



(a) Average Residual Workload (b) Fairness Index

Figure 11: DPRA With and Without Migration Overhead

get significantly improved throughput with 23.6%∼47.1% reduction on the total unprocessed workload than other distributed solutions. For the future work, we plan to make use of various task-filtering policies on the basis of our theory to solve more practical issues in load migration, e.g. by considering the situation that nodes may leave when receiving a migrated task. In addition, the flash crowd problem may happen to the system or some nodes as system scale is not that large, which will also be studied in the future.

Acknowledgments

This research is supported by a Hong Kong RGC grant HKU 7179/09E and a HKU Basic Research grant (Grant No. 10401460), and also in part by a Hong Kong UGC Special Equipment Grant (SEG HKU09).

Appendix

To get the expected overall balanced load level distribution in P2P Grid systems, the load amount to be shifted from every heavily loaded node p_i *should* *be equal to* $(\varphi_i^{(j)} - E^{(j)}(\Phi)) \cdot c_i - \theta \cdot c_i^2$*, where* $\theta = \frac{\sum_{p_k \in as(p_i)} l_k - E^{(j)}(\Phi) \cdot \sum_{p_k \in as(p_i)} c_k}{\sum_{p_k \in as(p_i)} c_k^2}$ *and* *as(*p_i*) refers to the acquaintance set of* p_i*.*

PROOF. According to *Theorem* 1, *Corollary* 1 and *Corollary* 2 in Section 4, our objective is to make the standard deviation of load level as small as possible. That is, the objective is to get the minimum $\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\varphi_i^{(j)} - E^{(j)}(\Phi))^2}$, where $E^{(j)}(\Phi)$ is the mathematical expectation at jth epoch. Based on the low-cost gossip protocol and locality property of network, the goal of each peer should switch to minimizing Δ (or Δ^2) subject to the constraint (20), where L' is a constant relative to the current epoch and $as(p_i)$ is the set of p_i's stored multi-hop gossiped acquaintance peers.

$$\Delta = \sqrt{\frac{\sum\limits_{p_k \in ns(p_i)} (\varphi_k^{(j)} - E^{(j)}(\Phi))^2}{|as(p_i)|}} \tag{19}$$

$$\sum_{p_k \in as(p_i)} l_k = L' \tag{20}$$

Since $\frac{\partial^2(\Delta^2)}{\partial l_k^2} > 0$, Δ^2 is convex with a minimum extreme point. Without loss of generality, for any node p_i, $|as(p_i)|$ is fixed within one epoch, then the target *Lagrange* function can be defined as Equation (21) and λ is the corresponding Lagrange multiplier.

$$F(as(p_i), \lambda) = \sum_{p_k \in ns(p_i)} (\tfrac{l_k}{c_k} - E^{(j)}(\Phi))^2 - \lambda(\sum_{p_k \in ns(p_i)} l_k - L') \tag{21}$$

Hence, Δ would get the minimum point iff Formula (22) can be met.

$$\frac{\partial F}{\partial l_k} = 0, where\ l_k\ is\ the\ load\ of\ p_k \in as(p_i) \tag{22}$$

Therefore, we can get the sufficient and necessary condition as follows, where k is the index of $p_k \in as(p_i)$

$$2(\tfrac{l_k}{c_k} - E^{(j)}(\Phi)) \cdot \tfrac{1}{c_k} = \lambda \tag{23}$$

Then, l_k is expected to be $c_k \cdot E^{(j)}(\Phi) + c_k^2 \cdot \frac{\lambda}{2}$. On the other hand, we denote $\theta = \frac{\lambda}{2}$, then θ can be calculated as: $\theta = \frac{l_k - c_k \cdot E^{(j)}(\Phi)}{c_k^2}$, where k is the index of

$p_k \in as(p_i)$. Hence, $\theta = \frac{\sum_{p_k \in as(p_i)} l_k - E^{(j)}(\Phi) \cdot \sum_{p_k \in as(p_i)} c_k}{\sum_{p_k \in as(p_i)} c_k^2}$ and the expected amount of load to be removed from each heavily loaded peer p_i is $\varphi_i^{(original)} \cdot c_i - l_i^{(expected)}$ $= \varphi_i^{(j)} \cdot c_i - (c_i \cdot E^{(j)}(\Phi) + c_i^2 \cdot \theta) = (\varphi_k^{(j)} - E^{(j)}(\Phi)) \cdot c_i - \theta \cdot c_i^2$
where $\varphi_i^{(original)}$ and $l_i^{(expected)}$ are referred to as the original load level on node p_i and its expected workload after the load migration, respectively. □

Remark: If p_i's acquaintance set contains all the participating peers or $(\sum_{p_k \in as(p_i)} l_k)$ / $(\sum_{p_k \in as(p_i)} c_k)$ = $E^{(j)}(\Phi)$, then, θ=0, which means the expected target load level is right $E^{(j)}(\Phi)$ and this is an ideal case. If $(\sum_{p_k \in as(p_i)} l_k)$ / $(\sum_{p_k \in as(p_i)} c_k) > E^{(j)}(\Phi)$, this indicates that p_i is located in a hotspot network area, in which most of the adjacent peers are heavily loaded, and vice versa.

References

[1] H. Abbes, C. Cerin, M. Jemni, Bonjourgrid: Orchestration of multi-instances of grid middlewares on institutional desktop grids, in: IEEE International Symposium on Parallel & Distributed Processing, IEEE, 2009, pp. 1–8.

[2] A. R. Butt, R. Zhang, C. Y. Hu, A self-organizing flock of condors, Journal of Parallel and Distributed Computing 66 (1) (2006) 145–161.

[3] J. Cao, F. B. Liu, C. Z. Xu, P2pgrid: integrating p2p networks into the grid environment: Research articles, Vol. 19, John Wiley and Sons Ltd., Chichester, UK, 2007, pp. 1023–1046.

[4] A. Luther, R. Buyya, R. Ranjan, S. Venugopal, Alchemi: A .net-based grid computing framework and its integration into global grids, Tech. rep., Grid Computing and Distributed Systems Laboratory, Univ. of Melbourne (Dec. 2003).

[5] H. Abbes, C. Cérin, M. Jemni, Pastrygrid: decentralisation of the execution of distributed applications in desktop grid, in: MGC '08: Proceedings of the 6th international workshop on Middleware for grid computing, ACM, New York, NY, USA, 2008, pp. 1–6.

[6] E. Byun, et al., Self-gridron: Reliable, autonomous, and fully decentralized desktop grid computing system based on neural overlay network, in: PDPTA'08: The International Conference on Parallel and Distributed Processing Techniques and Applications, 2008, pp. 569–575.

[7] V. K. Naik, S. Sivasubramanian, D. Bantz, S. Krishnan, Harmony: a desktop

grid for delivering enterprise computations, in: 4th International Workshop on Grid Computing, 2003.

[8] H. Zhao, X. Liu, X. Li, A taxonomy of peer-to-peer desktop grid paradigms, Cluster Computing (2010) 1–16.

[9] S. Di, C.-L. Wang, D. H. Hu, Gossip-based dynamic load balancing in a self-organized desktop grid, in: HPCAsia '09: Proceedings of the 10th High-Performance Computing Asia, 2009, pp. 85–92.

[10] J.-H. Hyun, An effective scheduling method for more reliable execution on desktop grids (2010) 172–179.

[11] G. Aggarwal, R. Motwani, A. Zhu, The load rebalancing problem, in: SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures, New York, NY, USA, 2003, pp. 258–265.

[12] C.-H. Hsu, T.-L. Chen, J.-H. Park, On improving resource utilization and system throughput of master slave job scheduling in heterogeneous systems, Journal of Supercomputing 45 (2008) 129–150.

[13] T. Repantis, Y. Drougas, V. Kalogeraki, Adaptive component composition and load balancing for distributed stream processing applications, Peer-to-Peer Networking and Applications 2 (1) (2009) 60–74.

[14] Y. Drougas, V. Kalogeraki, A fair resource allocation algorithm for peer-to-peer overlays, INFOCOM'05: 24th Annual Joint Conference of the IEEE Computer and Communications Societies 4 (2005) 2853–2858 vol. 4.

[15] G. Di Fatta, M. R. Berthold, Decentralized load balancing for highly irregular search problems, Microprocess. Microsyst. 31 (2007) 273–281.

[16] T. Repantis, Y. Drougas, V. Kalogeraki, Adaptive resource management in peer-to-peer middleware, in: Workshop on Parallel and Distributed Real-Time Systems (WPDRTS), in conjunction with 19th IPDPS, 2005, pp. 132–140.

[17] G. Huang, G. Wu, Z. Chen, A fair load balancing algorithm for hypercube-based dht networks, in: G. Dong, X. Lin, W. Wang, Y. Yang, J. Yu (Eds.), Advances in Data and Web Management, Vol. 4505 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Berlin, Heidelberg, 2007, Ch. 15, pp. 116–126.

[18] R. K. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling, John Wiley & Sons, 1991.

[19] T. Pitoura, P. Triantafillou, Load distribution fairness in p2p data management systems, in: 23th ICDE07: International Conference on Data Engineering, 2007, pp. 396–405.

[20] K. Eger, U. Killat, Fair resource allocation in peer-to-peer networks (extended version), Computer Communications 30 (16) (2007) 3046–3054.

[21] A. Allavena, A. Demers, J. E. Hopcroft, Correctness of a gossip based membership protocol, in: PODC '05: Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing, New York, NY, USA, 2005, pp. 292–301.

[22] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, Gossip algorithms: design, analysis and applications, in: INFOCOM'05: 24th Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 3, 2005, pp. 1653–1664.

[23] Y. Zhou, D.-M. Chiu, J. C. S. Lui, A simple model for chunk-scheduling strategies in p2p streaming, IEEE/ACM Transactions on Networking 19 (1) (2011) 42–54.

[24] Y. Wang, T. Fu, D. M. Chiu, Analysis of load balancing algorithms in p2p streaming, in: 46th Annual Allerton Conference on Communication, Control, and Computing, 2008, pp. 960–967.

[25] Brite topology generator: http://cs-pub.bu.edu/brite/.

[26] J. D. Sonnek, A. Chandra, J. B. Weissman, Adaptive reputation-based scheduling on unreliable distributed infrastructures, IEEE Trans. Parallel Distrib. Syst.(TPDS) 18 (11) (2007) 1551–1564.

[27] S. K. Kwan, J. K. Muppala, Resource discovery and scheduling in unstructured peer-to-peer desktop grids, International Conference on Parallel Processing Workshops (2010) 303–312.

[28] O. Sinnen, L. A. Sousa, Communication contention in task scheduling, IEEE Transactions on Parallel and Distributed Systems(TPDS) 16 (2005) 503–515.

[29] Y. F. Jeffrey, J. Chase, B. Chun, S. Schwab, A. Vahdat, Sharp: An architecture for secure resource peering, in: SOSP'03: Proceeding of 19th ACM Symposium on Operating System Principles, 2003, pp. 133–148.

[30] A. Benoit, M. Hakem, Y. Robert, Contention awareness and fault-tolerant scheduling for precedence constrained tasks in heterogeneous systems, Parallel Computing 35 (2) (2009) 83–108.

[31] P. Berenbrink, T. Friedetzky, L. A. Goldberg, P. Goldberg, Z. Hu, R. Martin, Distributed selfish load balancing, in: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, SODA '06, ACM, New York, NY, USA, 2006, pp. 354–363.

[32] U. Endriss, N. Maudet, F. Sadri, F. Toni, Negotiating socially optimal allocations of resources, Journal of Artificial Intelligence Research 25 (2006) 315–348.

[33] D. Grosu, A. T. Chronopoulos, M. Y. Leung, Cooperative load balancing in distributed systems, Concurr. Comput. : Pract. Exper. 20 (16) (2008) 1953–1976.

[34] J. F. Nash, Equilibrium points in n-person games, Proceedings of the National Academy of Sciences of the United States of America 36 (1) 48–49.

[35] M. Feldman, K. Lai, L. Zhang, The proportional-share allocation market for computational resources, IEEE Transactions on Parallel and Distributed Systems 20 (8) (2009) 1075–1088.

[36] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: SIGCOMM '01: Proceedings of the 2001 conference on App., tech., arch., and prot. for comp. comm., ACM, New York, NY, USA, 2001, pp. 161–172.

[37] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, in: SIGCOMM '01: Proceedings of the 2001 conference on App., tech., arch., and prot. for comp. comm., Vol. 31, ACM, New York, NY, USA, 2001, pp. 149–160.

[38] M. Jelasity, W. Kowalczyk, V. M. Steen, Newscast computing, Tech. rep., Vrije Universiteit Amsterdam (2006).

[39] M. Jelasity, A. Montresor, O. Babaoglu, T-man: Gossip-based fast overlay topology construction, Computer Networks 53 (13) (2009) 2321–2339.

[40] J. B. Jimenez, D. Caromel, M. Leyton, J. M. Piquer, Load information sharing policies in communication-intensive parallel applications, in: 6th IEEE International Symposium and School on Advance Distributed Systems (ISSADS), 2006.

[41] H. Kameda, J. Li, C. Kim, Y. Zhang, Optimal Load Balancing in Distributed Computer Systems (Telecomm. Networks and Comp. Systems), Springer-Verlag Telos.

[42] R. Wolski, N. Spring, J. Hayes, Predicting the cpu availability of time-shared unix systems on the computational grid, in: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing, HPDC '99, IEEE Computer Society, Washington, DC, USA, 1999, pp. 105–112.

[43] P. A. Dinda, D. R. O'Hallaron, Host load prediction using linear models, Cluster Computing 3 (2000) 265–280.

[44] L. Carrington, A. Snavely, N. Wolter, A performance prediction framework for scientific applications, Future Gener. Comput. Syst. 22 (2006) 336–346.

[45] Y. Zhang, W. Sun, Y. Inoguchi, Predicting running time of grid tasks based on cpu load predictions, in: 7th International Conference on Grid Computing, 2006, pp. 286–292.

[46] J. Zhang, R. J. Figueiredo, Adaptive predictor integration for system performance prediction, IPDPS07: International Parallel and Distributed Processing Symposium 0 (2007) 87–96.

[47] L. Huang, J. Jia, B. Yu, B.-G. Chun, P. Maniatis, M. Naik, Predicting execution time of computer programs using sparse polynomial regression, in: In NIPS'10: 24th Annual Conference on Neural Information Processing Systems, 2010, pp. 1–9.

[48] S. Di, C.-L. Wang, Conflict-minimizing dynamic load balancing for p2p desktop grid, in: Grid'10: The 11th IEEE/ACM International Conference on Grid Computing, 2010, pp. 137–144.

[49] Peersim simulator: http://peersim.sourceforge.net.

[50] Y. Huang, T. Z. J. Fu, D. M. Chiu, J. C. S. Lui, C. Huang, Challenges, design and analysis of a large-scale p2p-vod system, in: Proceedings of the ACM SIGCOMM conference on Data communication, New York, NY, USA, 2008, pp. 375–388.

[51] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, et al., A comparison of eleven static heuristics for mapping a class of independent tasks onto hetero-

geneous distributed computing systems, J. Parallel Distrib. Comput. 61 (6) (2001) 810–837.

[52] B. Radunovic, J. Y. Le Boudec, A unified framework for max-min and min-max fairness with applications, IEEE Transactions on Networking 15 (5) (2007) 1073–1083.