Conflict-minimizing Dynamic Load Balancing for P2P Desktop Grid

Sheng Di, Cho-Li Wang
Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
{sdi, clwang}@cs.hku.hk

***Abstract*—Fully decentralized resource allocation for P2P desktop Grid allows each participating node to act as both resource provider and requester. The system performance indicators (including throughput, makespan, etc) are easily degraded by the unbalanced load distribution, which is probably caused by the fast-changing states of heterogeneous resources due to arbitrary task submissions. Although the cooperative load rebalancing methods can mitigate the problem, they are likely to introduce the contention on under-utilized resources with growing task arrival rates, leading to the sub-optimal load balancing efficacy. Our focus is on how to optimize load balancing status by taking into account minimizing the conflict of autonomic task migration decisions in P2P desktop Grid. Our load rebalancing process is modeled as a set of independent stochastic Bernoulli trials by letting each heavily loaded node push its surplus loads to its surrounding lightly loaded nodes. We proved that the surplus load amount should be shifted based on a proper ratio by considering decision conflicts and designed a novel load balancing algorithm with provably small decision conflict probability. We derived an upper-bound for this probability, which will be reduced down to about 2% under our algorithm. Finally, we validated via simulation that the system performance can be significantly improved accordingly.**

I. Introduction

Desktop Grid (or Volunteer Computing Grid) may provide powerful integrated computational resource by leveraging desktop computers located at the edge of the Internet. Such platforms (e.g. BOINC [1] and XtremWeb [2]) have made great contributions to scientific researches since 2000. However, they inevitably suffered more and more issues with increasing system scales and application demands recently. The most serious one is that with a large number of volunteer nodes to manage, their traditional central-control architecture (such as central task scheduling and central data upload/download), not only faces high management cost and low flexibility but also causes Single-Point-Of-Failure and bottleneck problems. Consequently, fully-centralized P2P Desktop Grid allowing each autonomic desktop computer to individually allocate resources as a scheduler has become a promising trend. It is no wonder that there already emerged quite a few corresponding projects, such as PastryGrid [3], BonjourGrid [4], Condor-Flock P2P [5], Self-Gridron [6], [7], etc.

Under this decentralized architecture, dynamic load balancing algorithm allows each node to periodically and autonomously balance the uneven load distribution. It is crucial for gaining high system throughput and reliability of the dynamic environment. The existing algorithms [8], [9], [10] usually prefer push-mode (i.e. every heavily loaded node is allowed to unilaterally push its relatively surplus tasks to the lightly loaded nodes), owing to its simplicity and quick adjustment to the state-changes of heavily loaded nodes (or hot spots). In order to get system-wide balanced load status, the agent on every heavily loaded node is likely to view the states of many other lightly loaded nodes (a.k.a. *acquaintance nodes* or *local-view cache*) via multiple hops of message routing. Since it is inapplicable to coordinate/negotiate the task migration decisions among these heavily loaded nodes with dynamically changing states, it is non-trivial to avoid the situation that the lightly loaded nodes are over-utilized by being imported superfluous workload from outside (a.k.a. "tragedy of the commons situation" [11]). Such a decision-conflict problem has been confirmed by K. Christodoulopoulos et al.'s experiment [12]. The study undertaken in [13] 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.

Our focus is on designing a conflict-minimizing load balancing algorithm which may effectively balance uneven workloads for the dynamic P2P desktop Grid without any negotiation/reconfirmaion support, thus improving system throughput. We designed a *decentralized Bernoulli model* in which each heavily loaded node randomly selects other lightly loaded nodes to effectively mitigate the task migration decision conflict. We theoretically prove that our algorithm can help making best-response decisions for any node based on its local-view cache in the sense that the mutual decision conflict probability is minimized. On the other hand, we theoretically derive an upper-bound of such a probability to be a function of the load amount every node tries to shift. Comparatively, many existing decentralized load balancing strategies (such as CAN based method [10]) could mitigate the load unbalancing level to a certain extent,

978-1-4244-9349-4/10/$26.00 © 2010 IEEE

but never explicitly control the side-effect caused by the simultaneous task migration conflict during the load balancing progress. Due to the state-of-the-art simplicity of our design, many other improvements (such as replica generation policy [14], [15]) could be easily extended from our approach.

The remainder of the paper is organized as follows. We first describe related works in Section II. In Section III, we present the system overview and formulate the research. In Section IV, we model the decentralized load balancing problem to be the federated stochastic model and analyze the probability of the decision conflicts in theory. In Section V, we propose our conflict-minimizing algorithm, namely stochastic proportional idle resource allocation (SPIRA), based on our theoretical analysis. The system performance will be evaluated in Section VI. We conclude and present future work in Section VII.

II. Related Work

Most of the solutions rely on the central collection of global information, thus they are unsuitable for the large-scale dynamic resource allocation. For instance, the load rebalancing policy introduced by G. Aggarwal, et al [16], adopted a PARTITION algorithm and theoretically proved it to be 1.5-approximation optimal algorithm based on the global information about any node and task. M. Stillwell, et. al. proposed another resource allocation algorithm [17] especially suitable for multiple virtual clusters, also in terms of the globally collected information.

There are also some fully distributed resource allocation methods supporting load balancing in P2P desktop Grids. Most of them [10], [18], [19] are designed based on Distributed Hash Table (DHT) [20], [21]. Although DHT performs outstandingly in searching information within predictable delay, its performance always highly relies on its stable structure and it is usually complex and costly to maintain the whole topology in extraordinarily dynamic environment. In addition, these works usually adopt the selfish best-response algorithms at the self-organizing nodes but did not explicitly discuss about decision conflict problem, probably leading to a sub-optimal load balancing level or extra delay in making load rebalancing decisions. In comparison, not only can our SPIRA algorithm autonomously mitigate decision conflict efficiently, but it is also based on unstructured and dynamic connection, which is suitable for broader applications and flexible demands.

Some unstructured P2P based fair resource allocation (or load balancing) strategies [9], [22] also emerged these years. Y. Drougas and V. Kalogeraki [9], for example, proposed an algorithm based on replication requests specifically suitable for P2P data sharing applications. Each over-utilized node will create a new copy on the lightest neighbor node for its over-requested pieces. This approach is applicable for P2P data sharing scenarios, but unsuitable for Grid computing ones, because the computing resource (such as CPU) can not be replicated at will. Moreover, decision making conflict problem is also ignorable in their solution.

III. System Overview

Assume there are n heterogeneous nodes in the system, denoted by p_i, each with capacity c_i, where $1 \leq i \leq n$. The capacity here means a processing speed of some service, such as CPU clock rate or other services' rate in processing requests. For each load balancing interval (or snapshot), we assume there are w_i independent tasks on p_i, denoted as t_{ik}, $1 \leq k \leq w_i$. t_{ik}'s load (i.e. the number of instructions to perform) is represented as l_{ik}. Then, the total load of node p_i can be calculated as $l_i = \sum_{k=1}^{w_i} l_{ik}$. We define *load factor* of p_i (denoted by $lf(p_i)$) in Equation (1) and the higher $lf(p_i)$ is, the busier p_i gets.

$$lf(p_i) = \frac{l_i}{c_i} \tag{1}$$

Henceforth, how to efficiently balance the load factor among all nodes can be regarded as our direct objective. We define the mean value of load factors estimated by p_i's agent to be $\overline{LF} = \frac{\sum_{p_j \in AS(p_i) \bigcup \{p_i\}} lf(p_j)}{|AS(p_i) \bigcup \{p_i\}|}$, where $AS(p_i)$ indicates node p_i's acquaintance node set. Then, the heavily loaded node (lightly loaded node) is defined as the node whose load factor is higher (lower) than $\overline{LF}$. Further more, for any lightly loaded node p_i compared to $\overline{LF}$, the value of its relatively idle resource amount divided by its capacity is defined as the idle load factor $ilf(p_i)$, as shown in Equation (2); likewise, any heavily loaded node p_i's surplus load factor $slf(p_i)$ is calculated by its surplus load amount divided by its capacity, shown in Formula (3).

$$ilf(p_i) = \frac{\overline{LF} \cdot c_i - l_i}{c_i} = \overline{LF} - lf(p_i) \tag{2}$$

$$slf(p_i) = \frac{l_i - \overline{LF} \cdot c_i}{c_i} = lf(p_i) - \overline{LF} \tag{3}$$

In brief, every lightly loaded node periodically sends out its state information while every heavily loaded node asynchronously schedules/migrates its surplus tasks outward. We present the skeleton design of our conflict-minimizing load balancing strategy as follows (Algorithm 1):

This algorithm is run on every node to periodically check if the current node is either lightly or heavily loaded compared to the average load factor. Line 3$\sim$7 is used to distributively diffuse the states based on Newscast protocol [23] if the current node is lightly loaded. Under Newscast protocol, within every periodic message propagation cycle, each node needs to randomly choose C new neighbor nodes from the union set by merging its own original neighbor nodes and all neighbor nodes of one of its neighbors, incurring a peer sampling effect [24] around the whole system. C here is a fixed fan-out degree usually set to be $\log(n)$. Many existing researches (such as [23]) have proven that such protocol may effectively aggregate information with high

Algorithm 1 Skeleton of conflict-minimizing load balancing

Notice: This algorithm is executed on each node p_i
Input: Acquainted lightly loaded node set ($LAS(p_i)$) and local tasks
Output: load reassignment with minimal conflict probability

```
1: k = d. /*k is used to control the scheduling interval*/
2: while (TRUE) do
3:    if (lf(p_i)<LF) then
4:       Choose C(=log(n)) neighbors under Newscast protocol [23].
5:       Send its current state lf(p_i) to the C nodes.
6:       k = k − 1.
7:    end if
8:    if (k==0) then
9:       if (lf(p_i)>LF) then
10:         Perform SPIRA algorithm. /*Load balancing step*/
11:      end if
12:      k = d.
13:   end if
14:   Wait for a tiny period (i.e. a message propagation cycle);
15: end while
```

robustness even in dynamic environment. The evaluation of the message propagation protocols is beyond the scope of our research. At the end of each asynchronous scheduling interval (containing d message propagation cycles), the heavily loaded nodes will perform load balancing step based on our designed SPIRA algorithm which is described later. The SPIRA algorithm mainly focuses on three key issues: (1) How much load should be shifted? (2) Which lightly loaded nodes should be selected as the migration target? (3) which tasks should be migrated?

IV. Conflict-minimizing Stochastic Model

We leverage Bernoulli trial model to minimize the conflict of the task migrations by the autonomous heavily loaded nodes, achieving the approximated optimal load balancing status as well as high system throughput.

A. Decentralized Probabilistic Model

On any heavily loaded node p_i, the load balancing agent will independently select target lightly loaded nodes based on a united probabilistic model. We argue that the procedure of such asynchronously selecting target under-utilized nodes by every heavily loaded node in the competitive circumstance could be regarded as a set of Bernoulli trials, each of which in the theory of probability is defined as an experiment whose outcome is random and can be either of two possible results, "success" or "failure".

No doubt that any heavily loaded node p_i is expected to banish a certain amount of surplus load outward, however, their task migration decisions are likely to conflict with others' because each node is likely to view widely distributed node status for pursuing global load balancing objective. We define the *conflict decision* of one node to be an event that its load migration makes the target lightly loaded node become a new heavily loaded node, which is likely caused by the unknown aforehand occupation of other nodes' load migration with the same target. Thereby, the distributed load balancing procedure could be thought of as a Bernoulli trial model as follows: Collect redundant tasks from all heavily loaded nodes and gather all lightly loaded nodes respectively, as shown in Figure 1. Then throw all the surplus tasks randomly toward the lightly loaded nodes according to a probability-based rule, to see if the reschedule of some task migration will become a conflict decision. For example, in the case illustrated in Figure 1, all the surplus tasks will be migrated to lightly loaded nodes according to a uniform probability distribution, $\{\frac{3}{10},\frac{2}{10},\frac{3}{10},\frac{2}{10}\}$.

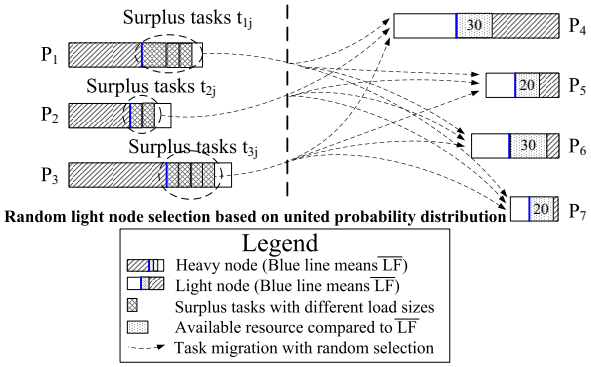


Figure 1. Decentralized load balancing can be viewed as Bernoulli trial

In terms of the Bernoulli trial model, we could prove the following lemma and theorems, which serve as the foundation of our algorithm shown in next section.

Lemma 1: **If** every heavily loaded node p_s banishes its surplus load (= slf(p_s)·c_s) and the probability of selecting the target lightly loaded node p_k from p_s's acquaintance nodes is $P(p_k)=\frac{ilf(p_k)\cdot c_k}{\sum_{pi\in LAS(p_s)} ilf(p_i)\cdot c_i}$ (where LAS(p_s)=$\{p_j|p_j\in$AS(p_s) and p_j is a lightly loaded node$\}$ each with a quite large number of elements), **then** the mathematical expectation of every node's load factor via such scheduling will approach $\overline{LF}$.

Proof. We first consider the case where all the tasks are of the same-size load and then extend it to heterogeneous task situation.

(1) In the situation where all tasks have the same load size μ, such load size can also be defined as one resource unit. Due to the large size of LAS(p_s) with randomly sampled nodes, every node p_s could estimate the $\overline{LF}$ accurately. Thus, the sum of surplus load amount is close to the sum of idle load amount, i.e. $\sum_{p_i\in LS} ilf(p_i)\cdot c_i$ = $\sum_{p_i\in HS} slf(p_i)\cdot c_i$ = $m\cdot\mu$, where *LS* and *HS* are the whole set of lightly loaded nodes and heavily loaded nodes respectively, and m is the total number of surplus tasks. Therefore, if we let each heavily loaded node select underloaded nodes based on the distributed Bernoulli model, the probability of x out of m tasks being allocated to a node p_k (i.e. probability mass function) can be calculated as Equation

(4), conforming to Binomial distribution.

$$P_{m\to p_k}(X=x) = C_m^x(P(p_k))^x(1-P(p_k))^{m-x} \quad (4)$$

Since each node could view masses of lightly loaded nodes, we assume $\sum_{p_i\in LAS(p_s)} ilf(p_i)\cdot c_i \approx \sum_{p_i\in LS} ilf(p_i)\cdot c_i$ (More general situations are discussed later). Hence, the mathematical expectation of the load amount allocated to the lightly loaded node p_k is E(p_k's received load) = $\mu\cdot(m\cdot P(p_k)) \approx \mu\cdot m\cdot(\frac{ilf(p_k)\cdot c_k}{\sum_{pi\in LS(p_k)} ilf(p_i)\cdot c_i}) = m\cdot(\frac{ilf(p_k)\cdot c_k}{(\sum_{pi\in HS(p_k)} slf(p_i)\cdot c_i)/\mu}) = m\cdot\frac{ilf(p_k)\cdot c_k}{m} = ilf(p_k)\cdot c_k$.

For any lightly loaded node p_k, since c_k and $lf(p_k)$ could be regarded to be two constants w.r.t. $ilf(p_k)$, the mathematical expectation E(updated $lf(p_k)$)=$\frac{E(p_k's\ received\ load)}{c_k}$=$lf(p_k)$ = $ilf(p_k)$+$lf(p_k)$=$\overline{LF}$.

For any heavily loaded node p_s, its load factor will also become $\overline{LF}$=$lf(p_s)-\frac{slf(p_s)\cdot c_s}{c_s}$.

(2) In the situation where the tasks are of different sizes, let us assume all tasks can be classified into F categories based on similar sizes: $\{m_i,\mu_i\}$, where $\sum_{i=1}^{F}\mu_i m_i = \sum_{p_i\in LS} ilf(p_i)\cdot c_i = \sum_{p_i\in HS} slf(p_i)\cdot c_i$. Then E(p_k's received load)=$\sum_{i=1}^{F}(\mu_i m_i P(p_k))$=$P(p_k)\sum_{i=1}^{F}\mu_i m_i$ = $\frac{ilf(p_k)\cdot c_k}{\sum_{p_i\in LS} ilf(p_i)\cdot c_i}\sum_{i=1}^{F}\mu_i m_i$ = $ilf(p_k)\cdot c_k$. Thus, $E(updated\ lf(p_k))$ = $\frac{E(p_k's\ received\ load)}{c_k}$ + $lf(p_k)$ = $ilf(p_k)$ + $lf(p_k)$ = $\overline{LF}$, so do all the heavily loaded nodes.■

Remark: In the situation where each node can view the status of all other nodes (i.e. $\sum_{p_i\in LAS(p_k)} ilf(p_i)\cdot c_i$=$\sum_{p_i\in LS} ilf(p_i)\cdot c_i$), the ideal load balancing status can be reached. In most of other realistic cases, however, $\sum_{p_i\in LAS(p_k)} ilf(p_i)\cdot c_i$ may not exactly approach $\sum_{p_i\in LS} ilf(p_i)\cdot c_i$, then the load balancing can only get an approximated result. For instance, in the worst case that each heavily loaded node just knows one lightly loaded node, then the load balancing will be gracefully degraded to the simple swap algorithm. For another example, if the whole node set is partitioned to several groups and any node may view the status of all other nodes in the same group, the load factor within one group may get ideal balanced result with high probability.

Theorem 1: The best-response strategy of shifting one heavily loaded node's surplus workload with the minimal decision conflict is migrating its surplus tasks toward its known lightly loaded nodes based on the probability which is proportional to the idle resource amount of lightly loaded nodes (i.e. $P(p_k)$=$\frac{ilf(p_k)\cdot c_k}{\sum_{pi\in LAS(p_k)} ilf(p_i)\cdot c_i}$).

Proof. Based on Lemma 1 and the *large number law*, the overall resource allocation is prone to be of high load balancing status when the scale of system and the number of tasks are large. Since every node's load factor will approach $\overline{LF}$, there will be no decision conflict in such balanced status. If one heavily loaded node changes its strategy of selecting lightly loaded nodes to deviate against the proportional idle resource amount, then all other heavily loaded nodes will also do the same deviations in that we assume every node adopts a uniform selection mechanism. Thereby, the deviation will be much larger than expected and may probably cause decision conflict on some specific node according to its definition. Hence, the conclusion follows. ■

We call the load balancing policy of each heavily loaded node selecting lightly loaded nodes with the probability proportional to their relatively idle resource amount *stochastic proportional idle resource allocation* (SPIRA).

B. Probability Analysis

In this section, we will calculate the approximated upper-bound of the probability of decision conflict when using the SPIRA method mentioned above.

Theorem 2: **Given** that every heavily loaded node moves its surplus load to its acquainted lightly loaded nodes selected by SPIRA method, **then** the probability of the decision conflict event on lightly loaded node p_k (denoted by $P_{DC}(p_k)$) is close to or less than $\frac{(xP(p_k))^{L_k+1}}{(L_k+1)!}$, as shown in Formula (5), where L_k=$m\cdot P(p_k)$, x indicates the total number of tasks to be shifted from all heavily loaded nodes and $\prec$ stands for "approaches or smaller than".

$$P_{DC}(p_k) \approx 1 - e^{-xP(p_k)}\sum_{i=0}^{L_k}\frac{1}{i!}(xP(p_k))^i \prec \frac{(x\cdot P(p_k))^{L_k+1}}{(L_k+1)!} \quad (5)$$

Proof. We denote the probability that there is no decision conflict event on the lightly loaded node p_k as $P_{\overline{DC}}(p_k)$, then $P_{DC}(p_k)$=1−$P_{\overline{DC}}(p_k)$. Since SPIRA is based on the Bernoulli trial model which conforms to Binomial distribution, we can get Formula (6) according to Formula (4), where L_k=$\frac{ilf(p_k)\cdot c_k}{\mu}$ and x is the total number of tasks to be shifted from all heavily loaded nodes. For simplicity, we consider the relatively serious situation where $\sum_{p_i\in LAS(p_s)} ilf(p_i)\cdot c_i \approx \sum_{p_i\in LS} ilf(p_i)\cdot c_i$, thus, $m\cdot P(p_k)$ = $m\cdot\frac{ilf(p_k)\cdot c_k}{\sum_{p_i\in LAS(p_s)} ilf(p_i)\cdot c_i}$ = $\frac{ilf(p_k)\cdot c_k}{(\sum_{p_i\in LS(p_k)} ilf(p_i)\cdot ci)/m}$ = $\frac{ilf(p_k)\cdot c_k}{\mu}$ = L_k. Intuitively, Formula (6) means the probability that the lightly loaded node p_k's received load amount is no greater than its tolerable threshold, $\overline{LF}$.

$$P_{\overline{DC}}(p_k) = \sum_{i=0}^{L_k} C_x^i(P(p_k))^i(1-P(p_k))^{x-i} \quad (6)$$

Without loss of generality, x is very large (i.e. $x \gg L_k \geq i$, e.g. $x = \frac{m}{2}$) and every $P(p_k)$ is relatively tiny due to the huge system scale, i.e. vast nodes and tasks. Thereby, we could get Formula (7) (Poisson's theorem) and Formula (8).

$$(1-P(p_k))^{x-i} \approx e^{-x\cdot P(p_k)} \quad (7)$$

$$C_x^i = \frac{x!}{i!(x-i)!} = \frac{x(x-1)(x-2)...(x-i+1)}{i!} \approx \frac{x^i}{i!} \quad (8)$$

Table I
THE ESTIMATED PROBABILITY OF CONFLICT EVENTS OUT OF 9870 LOAD MIGRATIONS TO 140 LIGHTLY LOADED 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}

Thus, we can convert Formula (6) to Formula (9).

$$P_{\overline{DC}}(p_k) \approx \sum_{i=0}^{L_k} \frac{1}{i!}(xP(p_k))^i e^{-xP(p_k)} \tag{9}$$

Accordingly, we will get the following deductions:

$$\begin{aligned}
&P_{DC}(p_k) = 1 - P_{\overline{DC}}(p_k) \\
&\approx 1 - \sum_{i=0}^{L_k} \tfrac{1}{i!}(xP(p_k))^i e^{-xP(p_k)} \\
&= 1 - e^{-xP(p_k)} \sum_{i=0}^{L_k} \tfrac{1}{i!}(xP(p_k))^i \ \ (Taylor's\ \ theorem) \\
&= 1 - e^{-xP(p_k)}(e^{xP(p_k)} - \tfrac{(xP(p_k))^{L_k+1}}{(L_k+1)!}e^{\xi}) \\
&= \tfrac{(xP(p_k))^{L_k+1}}{(L_k+1)!}e^{\xi - xP(p_k)}, \text{where } \ 0 < \xi < xP(p_k) \\
&< \tfrac{(xP(p_k))^{L_k+1}}{(L_k+1)!}
\end{aligned}$$

Thereby, $P_{DC}(p_k) \prec \frac{(xP(p_k))^{L_k+1}}{(L_k+1)!}$ ■

Based on the Formula (5), it is obvious that different x will get different probability on decision conflict event. The value of x is dependent on the load amount every heavily loaded node wants to dispel. If each heavily loaded node p_s dispels $\theta \cdot slf(p_s) \cdot c_s$ (where $0<\theta\leq 1$), then $x=\theta \cdot m$. As a use case, when $\theta=\frac{1}{3}$ (i.e. $x=\frac{m}{3}$) and node p_k's L_k=10, $P_{DC}(p_k) \prec \frac{(10/3)^{11}}{11!} \approx 1.41\%$.

We can also estimate the overall conflict probability by calculating the overall conflict events (denoted by $E_{DC}(x)$) over the total number $x(=\theta m)$ of tasks to be migrated from all heavily loaded nodes, as shown in Formula (10).

$$E_{DC}(x) = x \sum_{p_k \in LS} P(p_k) \cdot P_{DC}(p_k) \tag{10}$$

The overall probability of decision conflict is very small according to Equation (10). Based on this formula, Table I presents the estimated probability of conflict events to contend for 140 heterogeneous lightly loaded nodes whose idle resource amount (i.e. $ilf(p_k) \cdot c_k$) range from 1 to 140, where the total number of imported/migrated tasks is $\theta \cdot m = \theta \cdot \sum_{i=1}^{140} i = \theta \cdot 9870$. We observe that the conflict probability is already down to 0.025 if each heavily loaded node migrates 80% surplus load outward. In next section, we will formally propose our SPIRA algorithm via psudo-code. In Section VI, we will evaluate the performance more comprehensively through simulation.

V. CONFLICT-MINIMIZING LOAD BALANCING SCHEME

We further improve Algorithm 1 based on the above probability analysis in Algorithm 2:

Algorithm 2 SPIRA algorithm

This algorithm is executed on each heavily loaded node p_s
Input: Acquainted lightly loaded node set (*LAS*(p_s)) and local tasks
Output: Reschedule load with the minimum probability of decision conflict

1: Compute the selection probability distribution Δ for the lightly loaded acquaintance nodes; /*the probability of choosing lightly loaded node $p_k \propto ilf(p_k) \cdot c_k$.*/
2: *migLoad*=$\theta \cdot slf(p_s)$; /*$0<\theta\leq 1$*/
3: **while** (*migLoad*> 0) **do**
4: Select one lightly loaded node p_j based on distribution Δ.
5: Select one task t_{ik} according to *Pol*($S_f(S_p)$, R).
6: Migrate t_{ik} from p_s to p_j.
7: *migLoad*$-=l_{ik}$.
8: **end while**

Line 1 calculates the probability distribution Δ of selecting target lightly loaded nodes based on Lemma 1. Through line 2∼8, the target lightly loaded node will be selected as the node to execute the task migrated from the current node p_s according to Δ. Since different θ at Line 2 will introduce different conflict probability, the final allocation result may get distinct load balancing level. The exact value of θ could be determined based on Equation (5), Equation (10) or our simulation result presented in next section.

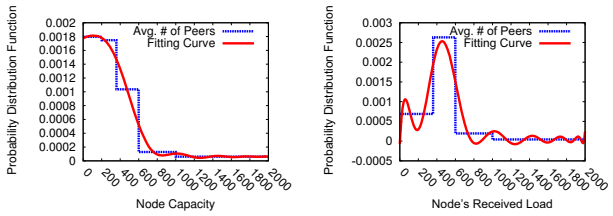
The tasks will be chosen based on some policy, denoted by *Pol*($S_f(S_p)$, R), where $S_f(S_p)$ is referred to as the *utility function set*, S_p refers to the set of all parameters needed, and R is the corresponding rule set providing constraints. The policy is mainly used to filter out the tasks violating the user's specific constraints. The detailed design of policy is able to be separated from our scheme, which is accordingly very flexible especially to dynamic application demands. As follows, we give a brief policy example on the transmission overhead constraint.

VI. PERFORMANCE EVALUATION

In this section, we analyze the SPIRA algorithm through the simulation over a large-scale fully distributed heterogeneous resource allocation test-bed.

A. Experimental Setting

We emulate a near-reality distributed resource sharing environment. We first construct an emulated physical network with n computers randomly connected with various bandwidths by *Waxman* model through Brite topology generator [25]. Then, we use PeerSim [26] to simulate the asynchronous events according to the polynomial fitting curve generated based on the average download/upload rate distribution reported in PPlive system [27], which serves as a typical Pareto distribution case to emulate the heterogeneous workload/capacity of participating nodes respectively. The distribution is shown as the red curve in Figure 2 and



(a) Capacity Distribution (b) Workload Distribution

Figure 2. Capacity and workload distribution among nodes

the total workload is set as equal to the total capacity. The neighbors of each node are dynamically changed over time yet the neighbor degree (=$\log(n)$) is always fixed. The nodes' status is delivered via the cycle-driven Newscast protocol [23] whose cycle length (i.e. the interval between two sequential messages on one node) is 5 minutes. The rescheduling interval at each node is 6 cycles, which is also the lifetime of each message.

The overhead of delivering messages can be neglected: Without loss of generality, suppose one message contains data payload of 80 bytes and the header information of 20 bytes, then each state message would only take about 100 bytes in total. Suppose the total number n of nodes is up to 10^6, then each lightly loaded node needs to communicate with other 20 nodes, and the total amount of data transmitted can be estimated as 20×100 *bytes*=$2K$ *bytes* per cycle, which is extremely small compared to the ordinary bandwidth (at least 1M*bytes*). On the other hand, the data transmission cost is also overlooked in our simulation due to two factors: (1) In our solution, the line 5 in Algorithm 2 has filtered all the tasks demanding super high transmission cost. That is, based on daily-life experiences, the IO-intensive tasks are not recommended to be rescheduled frequently unless thus will lead to distinct performance improvement. (2) In practical cases, due to the queuing model on each node, the overhead of migrating one task's data toward any particular node is likely to overlay with the execution of previous tasks queued on such node, actually suffering little task migration cost impact.

B. Simulation Result

We evaluate our conflict-minimizing solution (i.e. SPIRA algorithm) with different load sizes of tasks, respectively ranging from 2 through 64 GFloatpoints, as shown in Figure 3. This test is carried out via 1000 simulated heterogeneous nodes and migration ratio (θ)=0.9 (i.e. about 90% of surplus load on every heavily loaded node will be migrated outward). We can clearly see that the conflict probability (a.k.a. conflict event ratio which is calculated as the total number of conflict events divided by the total times of load migrations) decreases with the decreasing size of tasks. This is because the smaller size of task, the larger L_k will be in Formula (5), then the smaller conflict probability we will get according to the Theorem 2. The task load will randomly range from 2 to 32 in the following experiments.

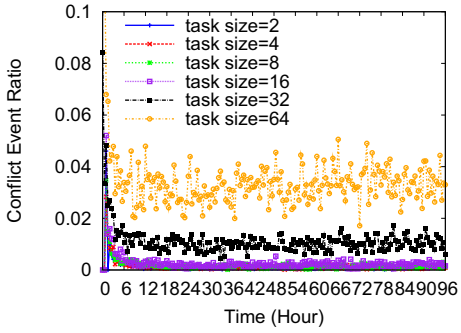
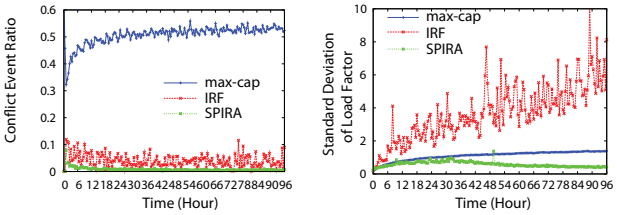


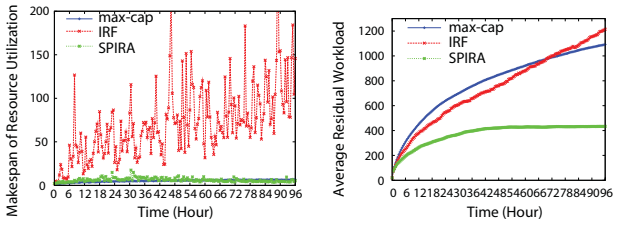
Figure 3. Conflict probability of SPIRA with various task sizes

The simulation result without any load balancing shows that the imbalanced workload caused by arbitrary task submissions will be significantly aggravated over time. Through the simulation of the four-days duration (96 hours), the final standard deviation of load factor will go up to 170 and the makespan (the highest load factor) will converge to 1590. Figure 4 (θ=1) presents the conflict probability, standard deviation of load factor, makespan, and average residual workload under different dynamic load balancing algorithms. Inspired by max-cap [28], we devised a max-cap based load balancing algorithm: Each heavily loaded node will periodically migrate the surplus tasks to other lightly loaded nodes based on the probability proportional to nodes' capacities. For Idlest Resource First (IRF), each node will schedule the surplus tasks onto its viewed idlest resource. Figure 4 indicates that the SPIRA algorithm will prominently reduce the conflict probability (Figure 4 (a)) and its performance (including load balancing level and total number of processed workloads) will always outperform the other two strategies (Figure 4 (b)).

As proved in Section IV.B, the lower migration ratio is, the lower decision conflict probability is. However, lower migration ratio does not mean 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 factor ($\overline{LF}$), either do the lightly loaded nodes. As a result, there must be a tradeoff by considering the load migration and the control over decision conflict. Figure 5 shows the load balancing result under different load migration ratio (i.e. θ). Figure 5 (a) shows that the decision conflicts are mitigated clearly with decreasing value of θ. The load balancing status (including standard deviation and makespan) shown in Figure 5 (b) and (c), however, gets the best effect when θ approaches 0.9 and very unstable when θ=1, as well as the system throughput which is reflected by the average residual workload (ARW) (smaller ARW means bigger throughput). As analyzed above, the θ=1 causing irregular result is due to the non-ignorable decision

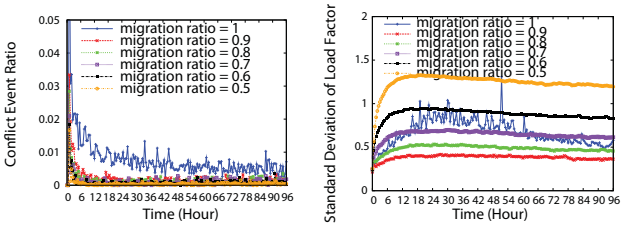


(a) Conflict Event Ratio (b) Standard Deviation of Load Factor

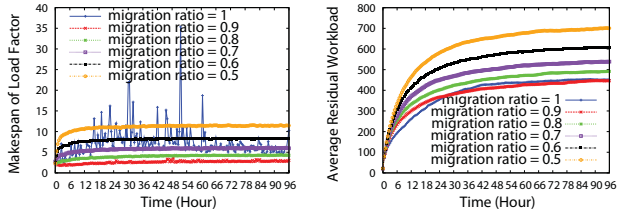


(c) Makespan (d) Average Residual Workload

Figure 4. Comparison among different load balancing strategies



(a) Conflict Event Ratio (b) Standard Deviation of load factor



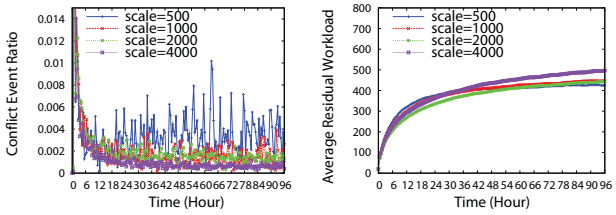
(c) Makespan (d) Average Residual Workload

Figure 5. SPIRA algorithm under different load migration ratios

conflict. Thus, $\theta \approx 0.9$ should be recommended in such fully decentralized competitive platform and we mainly adopt this value in the rest experiments.

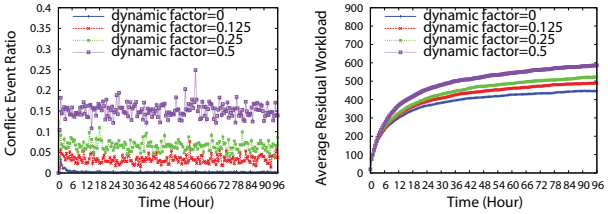
We present the scalability of SPIRA algorithm in Figure 6: The higher scalability of system, the lower conflict probability, but the system throughput (i.e. average residual workload) will not be influenced notably.

We note that the conflict probability is usually reduced down to 2% or even lower in the above stable cases under our SPIRA algorithm. Further more, we also evaluate our solution in dynamic situations, where each node may join/leave over time. In Figure 7, the dynamic factor α means the ratio of the nodes whose states are arbitrarily changed out of all nodes every half hour. For instance, α=0.5 indicates half nodes are disconnected and substituted by new ones.



(a) Conflict Event Ratio (b) Average Residual Workload

Figure 6. Scalability of SPIRA algorithm



(a) Conflict Event Ratio (b) Average Residual Workload

Figure 7. SPIRA algorithm over different dynamic environment

Figure 7 (a) shows the conflict probability is still restricted to 5% when the portion of arbitrarily changed nodes ranges from 12.5% $\sim$ 25%, which can be satisfied in most of cases.

VII. Conclusion and Future Work

In this paper, we designed a conflict-minimizing dynamic load balancing strategy suitable for fully decentralized resource allocation in P2P desktop Grid based on the Bernoulli trial model, with the state-of-the-art simplicity. We theoretically proved the upper-bound for the probability of conflicts among the autonomous load balancing decisions under our design should be very small. We also conclude that the migration ratio (i.e. the ratio of total migrated load amount and the total surplus load amount on heavily loaded nodes) is the key factor of impacting the decision conflict and system performance. Through simulation, we validated that our solution could significantly mitigate the unbalanced status of utilization load in dynamic fully decentralized environments, finally improving the overall performance immensely. In the future, we will further study how to leverage our algorithm to facilitate broader QoS, such as better fault tolerance ability.

Acknowledgments

This research is supported by Hong Kong RGC grant HKU 7179/09E and China 863 grant 2006AA01A111, and also in part by a Hong Kong UGC Special Equipment Grant (SEG HKU09)

References

[1] D. P. Anderson, "Boinc: a system for public-resource computing and storage," 2004, pp. 4–10.

[2] G. Fedak, C. Germain, V. Neri, and F. Cappello, "Xtremweb: a generic global computing system," in *CCGrid'01: Proceedings of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2001, pp. 582–587.

[3] H. Abbes, C. Cérin, and 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*. New York, NY, USA: ACM, 2008, pp. 1–6.

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

[5] A. R. Butt, R. Zhang, and C. Y. Hu, "A self-organizing flock of condors," *Journal of Parallel and Distributed Computing*, vol. 66, no. 1, pp. 145–161, January 2006.

[6] E. Byun, H. Kim, S. Choi, S. Lee, Y. S. Han, J.-M. Gil, and S. Y. Jung, "Self-gridron: Reliable, autonomous, and fully decentralized desktop grid computing system based on neural overlay network," in *PDPTA: The International Conference on Parallel and Distributed Processing Techniques and Applications*, 2008, pp. 569–575.

[7] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Alchemi: A .net-based enterprise grid computing system," in *International Conference on Internet Computing*, 2005, pp. 269–278.

[8] S. Di, C.-L. Wang, and 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.

[9] Y. Drougas and V. Kalogeraki, "A fair resource allocation algorithm for peer-to-peer overlays," pp. 2853–2858, 2005.

[10] J. S. Kim and et al., "Using content-addressable networks for load balancing in desktop grids," in *HPDC'07: 16th International Symposium on High Performance Distributed Computing*, New York, USA, 2007, pp. 189–198.

[11] "The tragedy of the commons," *Science*, vol. 162, no. 3859, pp. 1243–1248, December 1968.

[12] K. Christodoulopoulos, V. Sourlas, I. Mpakolas, and E. Varvarigos, "A comparison of centralized and distributed meta-scheduling architectures for computation and communication tasks in grid networks," *Comput. Commun.*, vol. 32, no. 7-10, pp. 1172–1184, May 2009.

[13] Y. F. Jeffrey, J. Chase, B. Chun, S. Schwab, and 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.

[14] C. Anglano and M. Canonico, "Scheduling algorithms for multiple bag-of-task applications on desktop grids: A knowledge-free approach," in *2008 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, April 2008, pp. 1–8.

[15] J. Wingstrom and H. Casanova, "Probabilistic allocation of tasks on desktop grids," in *IEEE International Symposium on Parallel & Distributed Processing,*, 2008, pp. 1–8.

[16] G. Aggarwal, R. Motwani, and A. Zhu, "The load rebalancing problem," in *SPAA '03: Proceedings of the 15th annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM, 2003, pp. 258–265.

[17] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation using virtual clusters," in *CCGRID '09: Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, vol. 0, Washington, DC, USA, 2009, pp. 260–267.

[18] R. Ranjan and R. Buyya, "Decentralized overlay for federation of enterprise clouds," *CoRR*, vol. abs/0811.2563, 2008.

[19] L. Cheng and et al., "Self-organising management overlays for future internet services," in *Modelling Autonomic Communications Environments*, ser. Lecture Notes in Computer Science, S. Meer, M. Burgess, and S. Denazis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, vol. 5276, ch. 7, pp. 74–89.

[20] S. Ratnasamy and et al., "A scalable content-addressable network," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 31, no. 4. New York, USA: ACM, October 2001, pp. 161–172.

[21] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: the SIGCOMM conference on Applications, tech., arch., and protocols for compu. commu.* ACM, 2001, pp. 149–160.

[22] K. Eger and U. Killat, "Fair resource allocation in peer-to-peer networks (extended version)," *Journal of Computer Communications*, vol. 30, no. 16, pp. 3046–3054, November 2007.

[23] M. Jelasity and M. van Steen, "Large-scale newscast computing on the internet," Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, Tech. Rep. IR-503, October 2002. [Online]. Available: http://citeseer.ist.psu.edu/jelasity02largescale.html

[24] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. v. Steen, "The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations," in *ACM/IFIP/USENIX 5th International Middleware*, 2004.

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

[26] "Peersim simulator: http://peersim.sourceforge.net."

[27] Y. Huang and et al., "Challenges, design and analysis of a large-scale p2p-vod system," in *SIGCOMM'08: the ACM SIGCOMM conference on Data communication*. New York, USA: ACM, 2008, pp. 375–388.

[28] M. Roussopoulos and M. Baker, "Practical load balancing for content requests in peer-to-peer networks," *Distributed Computing*, vol. 18, no. 6, pp. 421–434, June 2006.