Stability-Optimal Grouping Strategy of Peer-to-Peer Systems

Zhenhua Li [1], Jie Wu [2], Junfeng Xie [3], Tieying Zhang [4], Guihai Chen [3], and Yafei Dai [1]
[1] *Peking University, China*, [2] *Temple University, USA*,
[3] *Shanghai Jiaotong University, China*, [4] *Chinese Academy of Sciences, China*
lzh@net.pku.edu.cn, and jiewu@temple.edu

Abstract—When applied in high-churn Internet environments, P2P systems face a dilemma: although most participants are too unstable, a P2P system requires sufficient stable peers to provide satisfactory core services. Thus, determining how to leverage unstable nodes seems to be the only choice. Our primary idea is to group unstable nodes together in order to form an adequate number of stable service groups. Focusing on this topic, our main findings are three folds: 1) A general analytical model to investigate the grouping process of P2P systems is established, in which the stability-scalability tradeoff problem is paid special attention to; 2) We formalize the target of grouping as the *Maximum Stability Grouping* (MSG) problem. It proves to be not only NP-hard, but also infeasible; therefore, we restrict it to a feasible *Homogeneous MSG* (H-MSG) problem and deduce its optimal solution under the stochastic model; 3) We propose a homogeneous grouping strategy to fulfil the optimal solution. Comprehensive simulations have been performed on generated data sets and real-world traces from a P2P storage system and a P2P streaming system. Results show that our grouping strategy effectively captures the stability-scalability tradeoff: besides excellent stability, it gains much higher stable service capacity, with acceptable loss in scalability.

Index Terms—Peer-to-peer, stability, scalability, grouping, homogeneity, optimization

1 Introduction

Although P2P systems are famous for the accommodation and utilization of numerous unstable peers (called *dwarfs*), their core services rely heavily on stable peers (called *giants*), which have a large *session time length* (*stl*). For example, KaZaa [1] and eDonkey [2] employ stable *superpeers* for peer organizing, file indexing and searching, and BitTorrent [3] selects stable peers as the *trackers*. Besides, as a popular VoIP system, Skype [4] continuously picks out stable *super nodes* from its users to form the "backbone" of voice data streaming. Furthermore, measurements [5] show that in P2P video streaming systems, like PPLive [6], around 80% of the data traffic is delivered through less than 10% of the participants (mostly stable peers).

Previous studies [7]–[9] indicate that in real-world P2P systems, most participants are quite unstable (e.g., PCs and PDAs with session time length $<$ 60 minutes). So *we face a dilemma now: although most participants are too unstable to serve as stable peers, a P2P system requires sufficient stable peers to provide satisfactory core services.* Confronted with this dilemma, existing works can be briefly classified into the following three categories. (A detailed categorization is in Section 2 of the supplementary file. And the background and more related works can be found in Section 1 and Section 3 of the supplementary file.)

1) *GiantOnly.* Besides its broad use in unstructured P2P systems, the *GiantOnly* strategy is also employed in some DHT-based schemes, such as OpenDHT [10], where only giants can play the role of DHT nodes. Dwarfs are not allowed to enter the DHT, but are instead treated as clients.
2) *TotallyFlat.* Despite their substantial difference in overlay organization, Gnutella [11] and Chord [12] both construct a *Totally Flat* world for their participants. All peers are equal in function, no matter whether they are giants or dwarfs.
3) *StableNeighbor.* Since stable peers are usually deficient, some works try to detour this dilemma by grabbing more *Stable Neighbor*s for each peer. Godfrey et al. [8] focus on the issue of selecting a subset of the available node-set as relatively stable neighbors to replace failed ones. Yeung and Kowk [13] model the neighbor selection process as a cooperative game so that peers form stable coalitions with high possibilities.

The motivation of our work is based on the observation that in P2P systems, the capability of a single dwarf is negligible, but due to their overwhelming proportion, the dwarfs are still able to make significant contributions with their combined efforts — that is, combining several dwarfs to form a *stable service group* so as to act like a giant. A stable service group refers to a group of dwarfs/giants that cooperate to provide stable core services for the whole system. More importantly, the exploration of dwarf capability seems to be the *only* choice when attempting to offer satisfactory core services in high-churn scenarios with few giants. This highlights three requirements on our target: 1) We cannot compromise scalability for stability as GiantOnly; 2) We cannot compromise stability for scalability as TotallyFlat; 3) We do not want to play a zero-sum game as StableNeighbor, but try to provide fundamental optimizations or guarantees for the performance of core services.

In summation, the major question becomes: *in a P2P system with few giants and high churn, how can we group dwarfs in order to achieve significantly optimized stability, on the condition that the system scalability is guaranteed?*

Our primary idea is to admit all nodes and group *homogeneous* nodes together to form an adequate number of stable service groups (i.e., grouping dwarf with dwarf, giant with giant; we defer explaining the reason until Section 2.3). The inter- and intra-group connections are deployed distinctively, and the number of groups is deliberately tuned to guarantee the system scalability. Here, we use a simplified example depicted in Table 1 to illustrate our idea. By assigning more nodes to a dwarf group and fewer to a giant group, the dwarf group can survive for a time period equal to that of its giant counterpart, and thus we can get a sufficient number of stable service groups.

TABLE 1
Giant group vs. dwarf group

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Giant group																
n_1	•	•	•	•	•	•	•	•								
n_2							•	•	•	•	•	•	•	•	•	•
Dwarf group																
n_3	•	•	•	•	•											
n_4					$\downarrow$		•									
n_5					•	•	•									
n_6							$\downarrow$		•	•	•					
n_7							$\downarrow$		$\uparrow$		$\downarrow$				•	•
n_8						•	•	•	•		$\downarrow$				$\uparrow$	
n_9											•	•			$\uparrow$	
n_{10}												$\downarrow$		•	•	
n_{11}												•	•	•		

Note: 1) The black dot for node n_i at the j-th time slot indicates that n_i is online at that slot. 2) The vertical arrows imply the sequential relay of dwarfs, e.g., at the 5-th slot, n_3 is about to leave, and n_5 joins, thus the dwarf group still survives. In accordance with our definition of group stability, the dwarf group is stable throughout this period.

Our contributions are enumerated as follows:

- A general analytical model to investigate the grouping process of P2P systems is established, in which the stability-scalability tradeoff problem is paid special attention to.
- We formalize the target of grouping as the *Maximum Stability Grouping* (MSG) problem. Considering the intractability and infeasibility of MSG, we restrict it to a feasible *Homogeneous Maximum Stability Grouping* (H-MSG) problem and prove the optimal solution (it is also tractable) to H-MSG under the stochastic model. Both literatures and our measurements have indicated that the stochastic model holds.
- We propose a homogeneous grouping strategy to fulfil the optimal solution. Comprehensive simulations have been performed on generated data sets and real-world traces from a P2P storage system (AmazingStore [14]) and a P2P streaming system (CoolFish [15]). Results show that our grouping strategy effectively captures the stability-scalability tradeoff: Besides excellent stability, it gains much higher stable service capacity, with acceptable loss in scalability.

A final note is that we strongly feel our grouping model has its applicability in general distributed systems. In this model, most notations and the MSG/H-MSG problem also exist in a distributed system with heterogeneous members and high churn, and the optimal solution under the stochastic model mainly holds.

2 Grouping Model

2.1 Notations and Preliminaries

Consider a P2P system S with N nodes (each node has a probability to be online in a given *Period*). We want to group these N nodes into m disjoint groups: $G_1, G_2, \cdots, G_m$. T is the random variable of group session time length (*stl*), and τ_k denotes G_k's *stl*. We use Ψ as the random variable of group stability. A group G_k's stability ψ_k is mostly determined by its *stl* (τ_k). Basically, $\psi_k = \frac{\tau_k}{Period}$. The basic property of a group is that several nodes in this group can provide continuous and stable service for a period, so we want to make each ψ_k as high as possible. *Period* is usually set to 24 hours for a practical system.

A group G_k's service capability C_k is considered as the *time-weighted average* of its members' service capabilities rather than the *sum*, because all the members of G_k actually provide the same service functions as one single node. Grouping several dwarfs can just enhance their integrated stability, but cannot increase their integrated service capability. For example, four dwarfs in Fig. 1 are combined to form one stable group. The group acts like a giant with 533GB storage and 683Kbps (or 300Kbps) bandwidth. $533 = \frac{6}{24} \cdot 300 + \frac{4}{24} \cdot 100 + \frac{8}{24} \cdot 800 + \frac{6}{24} \cdot 700$, since in each overlapped period only the dwarf with the strongest capability online is in service. G_k's bandwidth is scenario-oriented: in a common scenario, it is calculated like G_k's storage; but in a bandwidth-sensitive scenario, it is $\min\{B_1, B_2, B_3, B_4\} = 300$Kbps, e.g., if G_k acts as a "backbone" super node in Skype, it can only report 300Kbps to the Skype system because any temporary shortage in bandwidth would cause voice streaming interruption. It should be noted that a group's service capability can be measured from different metrics according to specific application scenarios, e.g., bandwidth (in P2P media streaming systems), CPU/memory (in P2P computing systems), storage (in P2P storage systems), search efficiency (in general P2P systems), and so on. Detailed discussion on C_k is in Section 6.3 of the supplementary file.

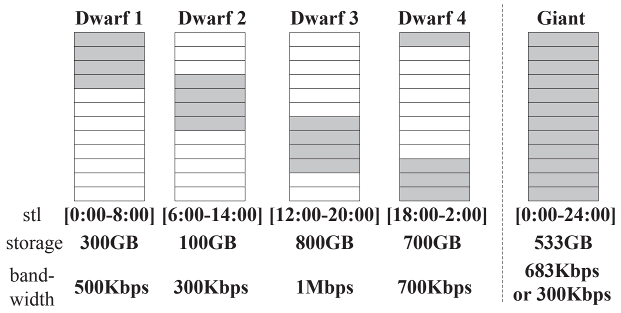

Fig. 1. A group of four dwarfs acts like a giant.

Table 2 is a reference of the basic notations used in this paper. Each of them will be exhaustively explained at their first appearance.

TABLE 2
Basic notations

Notation	Definition
$Period$	a given time period during which each node has a probability to be online. For a practical P2P system, $Period$ is usually 24 hours.
N	number of nodes (no matter whether online or offline).
m	number of groups.
G_k	the k-th group.
T, τ_k	T is the random variable of group session time length (*stl*), and τ_k is a group G_k's *stl*.
Ψ, ψ_k	Ψ is the random variable of group stability, and $\psi_1, \psi_2, \ldots, \psi_n$ are its sampling, i.e., ψ_k is the value of G_k's stability.
C, C_k	C is the random variable of group service capability, and C_k is a group G_k's service capability.
$D(x)$	CDF of X.
st, or t	a stable time slot, i.e., a slot when the system has entered a stable state.
$v_t(i)$	PDF of the number of node arrivals (i) at the t-th time slot. Node arrival means a node changes its state from offline to online.

The scalability of a P2P system S depends on 1) the number of groups m, and 2) the average service capability of all groups. It is formulated as:

$$Scalability(S) = m \cdot \mathbb{E}(C) = m \cdot \overline{C} = \sum_{k=1}^{m} C_k. \quad (1)$$

Obviously, $Scalability(S)$ is maximized only when $m = N$. In this extreme case, there exists no overlapped online time period among the members of a group. In fact, in this case, each group is a single node, so every node fully contributes its service capability to the system.

The stability of a P2P system S is somehow more complicated. Our grouping strategy makes the improvement in group stability *seem* like (in fact not) a zero-sum game. The only way for one group to become more stable is to grab some members from other groups, which jeopardizes their stability. Therefore, we need to equalize the stability levels across all groups to maximize the overall stability from a system perspective (see Section 6.4 of the supplementary file for better and easier understanding). As a result, if we define $Stability(S) = \overline{\Psi} = \frac{1}{m}\sum_{k=1}^{m} \psi_k$, the simple example depicted in Fig. 2 clearly indicates its irrationality. For the same participants P_1 - P_7 with stability 0.1 - 0.7, the two grouping schemes S_1 and S_2 both divide them into $m = 4$ groups. Table 3 presents the stability of S_1 and S_2 under the "Exclusive" and "Independent" conditions, respectively. Here, "Exclusive" means the members of a group are exclusive in session time, while "Independent" means they are independent in session time. For example, in the 3rd line of Table 3, $\psi_1(S_1) = 1 - (1 - 0.1) \cdot (1 - 0.7) = 0.73$.

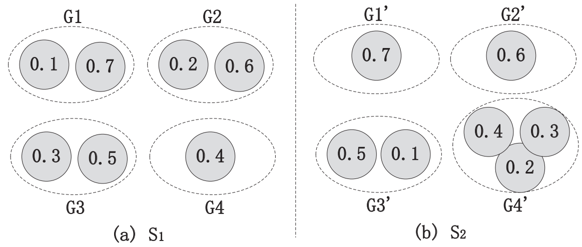

Fig. 2. Two grouping schemes S_1 and S_2 for the same participants.

TABLE 3
Stability of S_1 and S_2 in Fig. 2.

		ψ_1	ψ_2	ψ_3	ψ_4	$\overline{\Psi}$	$\mathbb{V}ar(\Psi)$
Exclusive	S_1	0.8	0.8	0.8	0.4	0.7	0.04
	S_2	0.7	0.6	0.6	0.9	0.7	0.02
Independent	S_1	0.73	0.68	0.65	0.4	0.615	0.0216
	S_2	0.7	0.6	0.55	0.664	0.6285	0.0044

The above example demonstrates that $Stability(S)$ depends mainly on $\mathbb{V}ar(\Psi)$, rather than $\overline{\Psi}$. In essence, what we want is an equalized system consisting of m groups with similar stability, rather than a polarized system where some groups are much more stable than others. Therefore, we define $Stability(S)$ as:

$$Stability(S) = \frac{1}{\mathbb{V}ar(\Psi)} = \frac{m-1}{\sum_{k=1}^{m}(\psi_k - \overline{\Psi})^2}. \quad (2)$$

Equations (1) and (2) put forward a scalability-stability tradeoff problem [16] for any grouping strategy. A small m leads to high stability because each group is composed of more dwarfs in average, and thus the stability is very high. However, a small m represents poor scalability because too few groups provide services. The discussion on a big m is alike. Therefore, our next step is to decide a proper m.

2.2 Condition: Guaranteed Scalability

For a P2P system S, in Section 2.1 we have defined $Scalability(S) = m \cdot \overline{C} = \sum_{k=1}^{m} C_k$. A group's service capability C_k can be measured from different metrics like bandwidth, CPU/memory, storage, search efficiency, and so on. Without loss of generality, here we use search efficiency as the metric of C_k since search efficiency is usually regarded as the most important (network-related) property of P2P systems. When each node of S sends a search request, the total message number with grouping must be *no more than* that without grouping. This can be formulated as a specific case of Equation (1): $Scalability(S) = \sum_{k=1}^{m} C_k = \sum_{k=1}^{m} (|G_k| \cdot \frac{1}{Avg\_search\_msg\#}) = N \cdot \frac{1}{Avg\_search\_msg\#}$.

Notably, we can still use bandwidth, CPU/memory, storage, etc. as the metric of C_k, and the corresponding discussion can be found in Section 6.3 of the supplementary file. Whatever metric we choose, *any grouping strategy has its fundamental condition: it must guarantee that its system scalability holds on the same level as the original system without grouping*.

As unstructured and DHT systems differ greatly in operation mechanism, their scalability guarantees are discussed separately below.

2.2.1 For unstructured systems

We take Gnutella as the representative of unstructured P2P systems. Consider a Gnutella network S_1 composed of N nodes with the average node degree = d and flooding search radius = TTL hops. If we group S_1 into a new system S_2, which is composed of m groups, most edges in S_1 would become inter-group edges in S_2, and the remaining edges in S_1 would become intra-group edges in S_2. This can cause two problems: 1) the average inter-group degree d_G is too large, and thus the groups of S_2 are over-densely connected; 2) the intra-group edges are too sparse, and thus a group may be disconnected. Therefore, we randomly trim the inter-group edges from S_2 (make sure S_2 is connected all along) until d_G is reduced close to d, so that S_2 has the same edge density as a common Gnutella network. Besides, for each group of S_2, we randomly add intra-group edges until this group is connected. A demo of the final state of S_2 is shown in Fig. 3:

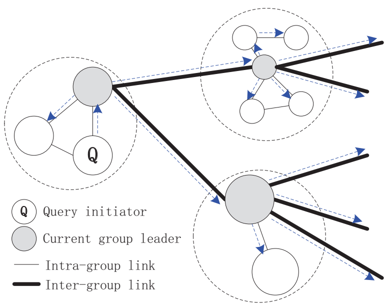

Fig. 3. A grouping demo of the unstructured P2P system. The dotted blue arrow illustrates the message flow of a search operation.

To guarantee the scalability of S_2, we make

$$Scalability(S_1) \leq Scalability(S_2),$$

that is $N \cdot \frac{1}{Avg\_search\_msg\#1} \leq N \cdot \frac{1}{Avg\_search\_msg\#2}$,

$$Avg\_search\_msg\#2 \leq Avg\_search\_msg\#1. \quad (3)$$

Suppose TTL' is the inter-group flooding radius of S_2. Inside a group G_k, the number of messages is almost $|G_k|$ because the intra-group flooding can usually reach all members. So, Equation (3) is transformed to

$$d_G{}^{TTL'} \cdot \frac{N}{m} \leq d^{TTL}. \quad (4)$$

Since $d_G \approx d$, Equation (4) is approximately

$$m \geq \frac{N}{d^{TTL-TTL'}}, \text{ or } \quad \frac{N}{m} \leq d^{TTL-TTL'}. \quad (5)$$

In the Gnutella network, usually d lies in 3 – 5 and $TTL \leq 7$. $TTL - TTL'$ may be 1, 2, or 3.

2.2.2 For DHT systems

We take Chord as the representative of DHT systems. Likewise, we group a Chord system S_1 into the new system S_2, which is composed of m groups. Since the members of a group share the same *ID* in DHT, for a group G_k, we randomly choose the *ID* of one member as the *ID* of G_k. The inter-group edges are organized in the same way as Chord. As mentioned in Section 2.2.1, for each group of S_2, we randomly add intra-group edges until this group is connected.

Equation (3) still holds for the grouping of DHT systems, but is formulated as

$$O(\log m) + \frac{N}{m} \leq O(\log N). \quad (6)$$

Equation (6) is a transcendental equation, so we just construct a feasible solution. Obviously, $m \in O(\frac{N}{logN})$ is one feasible solution, because

$$O(\log m) + \frac{N}{m} \in O(log\frac{N}{logN}) + O(logN)$$
$$\in O(logN) - O(\log\log N) + O(logN) \in O(logN).$$

In fact, $m \in O(\frac{N}{logN})$ means the average group size $\in O(logN)$.

2.3 Target: Maximum Stability Grouping Problem

We denote the set of nodes that will join in the system during a sufficiently long period by $\mathbb{S} = \{n_1, n_2, \ldots, n_L\}$. Assume each node n_i's join time $n_i.join$ and leave time $n_i.leave$ are *priori* knowledge (Of course this assumption is impractical, and we will address this problem later). The number of groups m is determined in Section 2.2. Our target is formalized as the following MSG problem:

Definition 1 (**Maximum Stability Grouping** Problem).

Instance: *A given* m, *and* $\mathbb{S} = \{n_1, n_2, \ldots, n_L\}$, *where each node* n_i*'s join time* $n_i.join$ *and leave time* $n_i.leave$ *are known.*

Solution: *A partition of* $\mathbb{S}$ *into* m *disjoint groups* $G_1, G_2, \ldots, G_m$.

Measure *to minimize:* $\mathbb{V}ar(\Psi) = \frac{1}{m-1} \sum_{k=1}^{m} (\psi_k - \overline{\Psi})^2$.

Then, we can prove the following theorem (the proof is in Section 4 of the supplementary file):

Theorem 1. *With a non-trivial* $m \geq 1$, *MSG is NP-hard.*

Besides intractability (NP-hard), MSG is also infeasible in that it entails the priori knowledge (i.e., prediction) of each node's join and leave time, which is impractical in real P2P systems. Thereby, we look into this issue from another perspective. Our approach deploys homogeneity more restrictively so as to reduce MSG into a feasible optimization problem, i.e, the *Homogeneous Maximum Stability Grouping* (H-MSG) problem, where only the distributions of *stl* - $D(.)$, and number of arrivals - $v.(.)$ need to be known. *We combine homogeneous nodes that have the same or similar* stl*s to form a group under the stochastic model*, that is to say, grouping dwarf with dwarf, giant with giant, and supposing the peers' churn (join, *stl*, etc.) mainly follows a stochastic process.

We believe it is invariably impossible for each group to involve one powerful giant because of its rarity in most high-churn scenarios. In this sense, the idea of grouping giants and dwarfs in a mixed way, is greatly invalidated. Such an idea will also induce low efficiency when the dramatic asymmetry in node capability is taken into account. For instance, bandwidth asymmetry prevents giant bandwidth from being fully leveraged when it communicates with dwarfs. And due to huge diversity in CPU/memory or storage, the departure of a giant may make it hard for the remaining dwarfs to take over its duty. For instance, no PC can take charge of a supercomputer in either computing or storage capability, except when these PCs cooperate in a quite specialized way. Besides, the homogeneous grouping strategy is the most efficient in bandwidth-sensitive scenarios (e.g., P2P media streaming), which would be explained in Section 6.3 of the supplementary file. Furthermore, the users (nodes) of each ISP usually exhibit certain homogeneity, especially in bandwidth, so our homogeneous grouping strategy has the potential to facilitate topology awareness, as well. In a word, it is both reasonable and efficient, at least in high-churn scenarios, to group nodes homogeneous in terms of *stl*s.

To solve the H-MSG problem, as shown in Fig. 4, the *stl* axis is divided into m intervals, i.e., $[y_0, y_1)$, $[y_1, y_2)$, …, $[y_{m-1}, y_m)$, where $y_0 = 0$ and $y_m = +\infty$, and the nodes whose *stl*s are in the same interval are destined to the same group. The target is to minimize $\mathbb{V}ar(\Psi)$. This solution seems to somehow jeopardize system stability by prohibiting any overlap of different *stl* ranges, but it should be reasonable, as mentioned above. The detailed design of our proposed grouping strategy is described in Section 6 of the supplementary file. And the distributed algorithm can be found in Section 9 of the supplementary file.

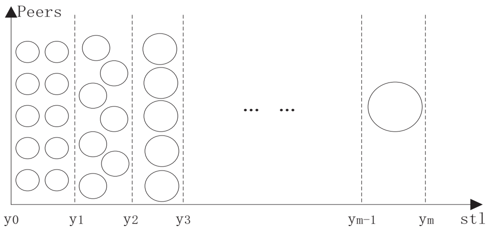

Fig. 4. Demo of the H-MSG problem.

2.4 Optimal Solution under the Stochastic Model

In this subsection, we first indicate that $D(.)$ and $v.(.)$ approximately follow a stochastic process though both literatures and our measurements, and then address the optimal solution of H-MSG.

2.4.1 Stochastic Model

It is widely assumed in literature that node arrival ($v.(.)$) is a memoryless and stochastic process: often a Poisson process [17]. Additionally, it is confirmed in [16] and [18] that the distribution of node *stl* ($D(.)$) exhibits a predictable stochastic pattern: often, but not always, a Zipf-like pattern [7], [9]. Generally, it is possible to figure out an approximate stochastic distribution of $D(.)$ and $v.(.)$ by monitoring node session history, although such information for a single node is hard to model. Below, we will show how to achieve this by taking the AmazingStore trace as an example. A brief description of the AmazingStore trace is in Section 7.2 of the supplementary file.

Fig. 5 and Fig. 6 illustrate that both the numbers of online users and joining users in AmazingStore exhibit obvious periodical distribution. Users behave very similarly at the same time every day. Although it is difficult to summarize Fig. 6 with a formula, we can easily approximate the stochastic distribution of $v.(.)$ through sampling and interpolation in Fig. 6.

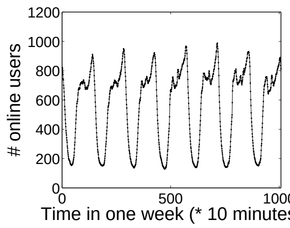

Fig. 5. Number of online users in AmazingStore.

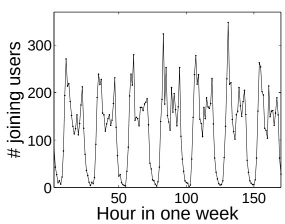

Fig. 6. Number of joining users in AmazingStore.

The session time pattern of all users in AmazingStore in three months is depicted in Fig. 7. This long-tail pattern deviates greatly from the well-known Zipf-like (or says power-law) distribution. Instead, the stretched exponential (SE) distribution [19] fits the session time pattern well. Thereby, the stochastic distribution of $D(.)$ is obtained.

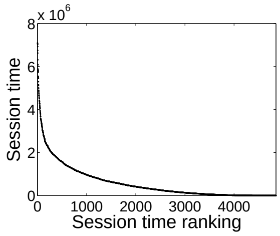

Fig. 7. Session time pattern of all users in AmazingStore in three months.

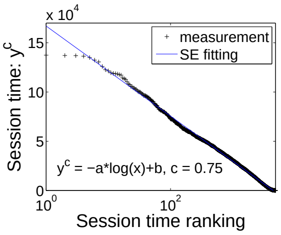

Fig. 8. Stretched Expotential distribution fits the session time pattern well.

Now we are sure that the session time of a node can be predicted by monitoring node session history. However, in an open P2P environment, such information for a single node is hard to get although the session time distribution of all nodes can be easily got. Then the problem is: how to estimate the session time of a node when it joins the system? As a matter of fact, it is impossible to accurately estimate such information when a new node joins, because we know nothing about it. Our solution is to estimate the session time of a new node as the average session time of existing nodes. As time goes, the information of a new node would be learnt, and then we can allocate it into a more proper group. Refer to Section 10 of the supplementary file for the performance evaluation.

2.4.2 Optimal Solution of H-MSG

To facilitate the analysis, we sample a slot st large enough so that the system size (i.e., the number of nodes online) is relatively stable (e.g., it slightly fluctuates around an estimated value) at that time. Then, the stability of a group G_k at slot st is

$$\psi_k = 1 - \mathbb{P}(\phi_k(st)), \tag{7}$$

where $\phi_k(st)$ denotes the event that G_k is empty at the st-th slot.

Theorem 2 indicates that the H-MSG problem is actually a feasible optimization problem, so long as $D(.)$ and $v.(.)$ follow a stochastic model.

Theorem 2. *ψ_k is the function of y_{k-1} and y_k.*

The proof is in Section 5 of the supplementary file.

Following Equations (2), (7), and Theorem 2, we can obtain Corollary 1:

Corollary 1. *The H-MSG problem can be reduced to a feasible optimization problem where $y_1, y_2, \ldots, y_{m-1}$ need to be determined to minimize $\mathbb{V}ar(\Psi)$, so long as $D(.)$ and $v.(.)$ follow a stochastic model.*

The above optimization problem can be calculated with the Matlab (version R2001a) nonlinear constrained optimization solver *fmincon(.)* and some other related solvers. Its computation complexity is polynomial for two reasons: Firstly, for the infinite summations $\sum_{i=0}^{+\infty}(\cdots)$, the upper bound $+\infty$ is in fact a limited (usually small) integer because the number of node arrivals in a time slot is limited. It is impossible that infinite nodes arrive at the P2P system in a time slot. Instead, usually at any time slot, there is at most one node joining in a group G_k. Secondly, *fmincon(.)* is implemented as a numerical algorithm with user-configured precision and number of iterations in Matlab, and thus its computation complexity is also polynomial. To sum up, we have the following conclusion:

Corollary 2. *The optimal solution to the H-MSG problem under the stochastic model is both feasible and tractable.*

3 Performance Evaluation

3.1 Environment Setup

Three data sets, including one generated data set and two real-world system traces, as described in Section 7 of the supplementary file, are used to evaluate the performance of our proposed grouping strategy.

3.2 Metrics

We evaluate our grouping strategy and the related works mainly from two aspects: stability and scalability. *Churn rate* is defined to measure stability. And we evaluate scalability from two orthogonal perspectives: *search efficiency* and *system storage capacity*. Additionally, we use *system stable storage* to measure the scalability of P2P storage systems (like AmazingStore), and *system stable bandwidth* to measure the scalability of bandwidth-sensitive P2P streaming systems (like CoolFish). Furthermore, we measure the *maintenance overhead* of related systems, using the generated data set. Finally, we evaluate the *load balance* situation of our proposed grouping strategy, using the AmazingStore trace. All the abovementioned metrics are elaborated on in Section 7.4 of the supplementary file.

3.3 Results on Generated Data Set

We first generate a demo data set with $N = 200$ nodes to illustrate how our grouping strategy works, with $m = \frac{N}{logN} = 26$. As shown in Fig. 9 and Fig. 10, in accordance with their *stl* intervals, all groups are sorted in ascending order and indexed accordingly (ID). Just as expected, the curve of the number of nodes in each group is skewed, which means that a dwarf group has to involve more nodes than its giant counterpart to maintain a comparable stability.

Then, we generate a data set with $N = 1000$ nodes and $m = 100$ groups. Fig. 11 demonstrates that, as we expected, TotallyFlat (Chord/Gnutella) is far

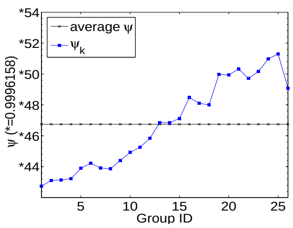

Fig. 9. Each group's stability and their mean value.

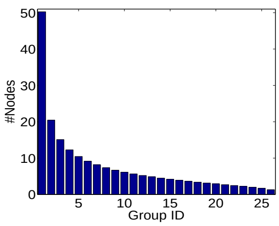

Fig. 10. Number of nodes in each group to gain the maximum stability.

more dynamic than Grouping. Out of our expectation, GiantOnly is also more dynamic than Grouping. Why do giants have more churns than our dwarf groups? The reason lies in that choosing $m = 100$ giants from $N = 1000$ nodes is too difficult when the node stl follows the exponential distribution (refer to Section 7.1 of the supplementary file). In fact, among the 100 "giants", most are not as stable as their dwarf-group counterparts, thus leading to our unexpected observation.

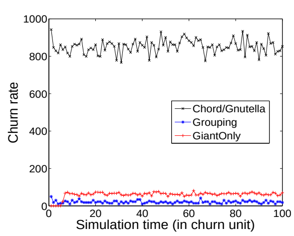

Fig. 11. Evolution of churn rate.

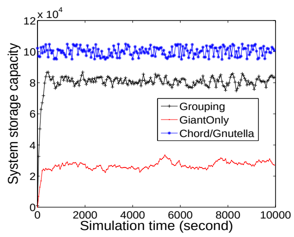

Fig. 12. Evolution of system storage capacity.

To contrast Grouping with Gnutella, Chord, and GiantOnly in search efficiency, we assume the target file locates on each group member uniformly. Let each node/group send a search query and record the average routing message number in Fig. 13. For Gnutella and Grouping-Gnutella, the bars show the message hop (TTL), while the tagged numbers in the error bars show the message number involved in a flooding search. For Chord, Grouping-Chord, and GiantOnly, the routing hop denotes the search message cost. They are consistent with our evaluations in Section 6.2 of the supplementary file.

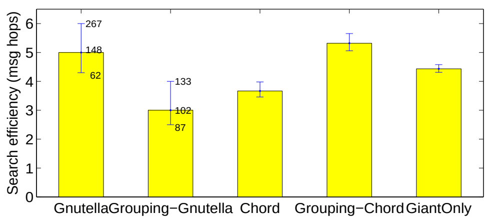

Fig. 13. Search efficiency comparison.

Fig. 12 explicates that Grouping's system storage capacity is perceivably greater than that of GiantOnly by almost three times. Nevertheless, it is still less than that of Gnutella/Chord by about 20%, which is the compensation that Grouping has to pay.

To compare Grouping with Gnutella, Chord, and GiantOnly as to maintenance overhead, we let the maintenance period be equal to the churn unit time (100 seconds). Their respective maintenance overheads are illustrated in Fig. 14. Gnutella and Chord only have inter-group overhead, where one node can be seen as one group. Obviously, Gnutella bears much more maintenance overhead than others. The inter-group overhead exceeds the intra-group overhead in Grouping-Gnutella, but the case is just opposite in Grouping-Chord. GiantOnly has less intra-group overhead, because each group member only sends its state to the group leader (a giant) in a period. The inter-group maintenance mechanism of GiantOnly is assumed to be TTL-flooding. Section 6.1 of the supplementary file contains the corresponding theoretical analysis.

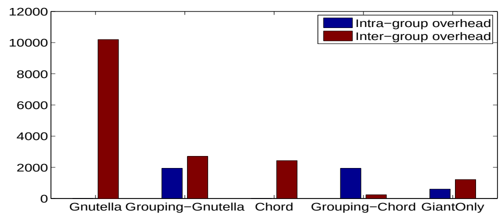

Fig. 14. Maintenance overhead comparison.

3.4 Results on AmazingStore Trace

As mentioned in Section 7.2 of the supplementary file, 4854 AmazingStore nodes are grouped into 396 groups, with the average group stability $\overline{\Psi} \approx 0.6$. $\overline{\Psi} \approx 0.6$ appears less than enough, but in fact is satisfactory when considering that all AmazingStore users come from China colleges. College students and teachers usually live a regular life, for example, being active from 8:00 – 22:00 and asleep from 23:00 – 7:00. AmazingStore always has much fewer users at night than in daytime, which can be proved from our real-time user log page [20].

Seen from Fig. 15 and Fig. 16, after grouping, both the churn rate and churn ratio of AmazingStore have greatly decreased. Even the highest churn ratio of AmazingStore-grouping is smaller than the lowest churn ratio of AmazingStore.

Fig. 17 shows that the system storage capacity of AmazingStore-grouping is less than that of AmazingStore, especially at the "hot" hours. During the other non-hot hours, they perform alike. As a P2P storage system, what is more important for AmazingStore is the stable storage capacity in Fig. 18. We change the stability ($\overline{\Psi}$) requirement to compare their stable storage capacities. Accordingly, when we choose a higher stability requirement, the number of groups m

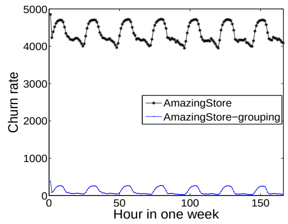

Fig. 15. Churn rates of AmazingStore.

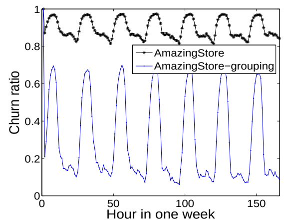

Fig. 16. Churn ratios of AmazingStore.

should be reduced. In all the four cases, AmazingStore possesses less than 30GB stable storage, which is much lower than that of AmazingStore-grouping.

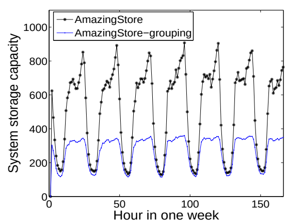

Fig. 17. System storage capacities of AmazingStore.

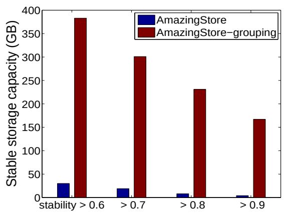

Fig. 18. Stable storage capacities of AmazingStore.

3.5 Results on CoolFish Trace

Since the CoolFish trace is divided into 9 sub-traces and each sub-trace is processed individually, we only depict the churn rates (in Fig. 19) and churn ratios (in Fig. 20) of the sub-trace on Apr. 13. The other 8 sub-traces are generally similar. In Fig. 20, there exist two exceptional churn times when no group is online. This mainly results from our server code updates.

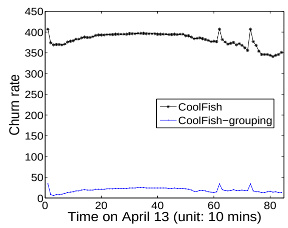

Fig. 19. Churn rates of CoolFish.

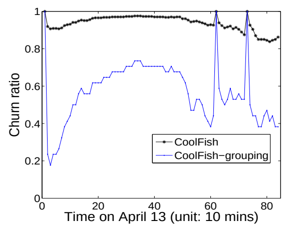

Fig. 20. Churn ratios of CoolFish.

System stable bandwidth is critical to P2P streaming systems because of their sensitivity to bandwidth vibration. Fig. 21 compares the system stable bandwidths of CoolFish and CoolFish-grouping each day. Here, "stable" means a group/node can provide stable bandwidth in more than 60% of time per day. Since CoolFish has few users at night, 60% is close to the ratio of daytime over a whole day. Clearly, the system stable bandwidth of CoolFish is trivial compared to that of CoolFish-grouping.

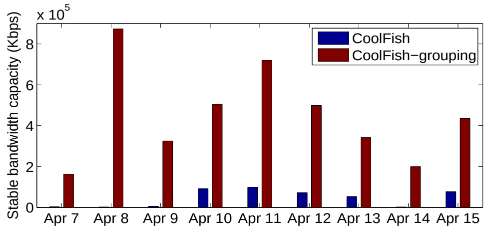

Fig. 21. Stable bandwidth capacities of CoolFish.

3.6 Results Summarization

Refer to Section 7.5 of the supplementary file.

4 Conclusion

Motivated by the dilemma of stable peers in P2P systems, in this paper we investigate how to group unstable nodes together in order to form sufficient stable service groups. A general grouping model is established and a homogeneous grouping strategy is proposed to acquire optimal stability with guaranteed scalability. Simulations on generated data sets and real-world traces reveal that our grouping strategy derives a better stability-scalability tradeoff: besides excellent stability, it gains much higher stable service capacity, with acceptable loss in scalability.

References

[1] "KaZaa web site," http://www.kazaa.com/.
[2] "eDonkey web site," http://www.emule-project.net/.
[3] "BitTorrent web site," http://www.bittorrent.com/.
[4] "Skype web site," http://www.skype.com/.
[5] F. Wang, J. Liu, and Y. Xiong, "Stable Peers: Existence, Importance, and Application in Peer-to-Peer Live Video Streaming," In INFOCOM '08, pp. 1364-1372, Phoenix, AZ, USA, Apr. 2008.
[6] "PPLive web site," http://www.pptv.com/.
[7] S. Saroiu, P.K. Gummadi, and S.D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," Multimedia Computing and Networking (MMCN '02), pp. 156-170, Jan. 2002.
[8] P. Godfrey, S. Shenker, and I. Stoica, "Minimizing churn in distributed systems," In SIGCOMM '06, pp. 147-158, Sep. 2006.
[9] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," In SOSP '03, pp. 314-329, Oct. 2003.
[10] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: a public DHT service and its uses," In SIGCOMM '05, pp. 73-84, Aug. 2005.
[11] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the Gnutella Network," IEEE Internet Computing, pp. 50-57, Jan. 2002.
[12] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," In SIGCOMM '01, pp. 149-160.
[13] M. Yeung, and Y. Kwok, "Game Theoretic Peer Selection for Resilient Peer-to-Peer Media Streaming Systems," In ICDCS '08.
[14] "AmazingStore web site," http://www.amazingstore.org/.
[15] "CoolFish web site," http://www.cool-fish.org/.
[16] C. Wang and K. Harfoush, "On the stability-scalability tradeoff of DHT deployment," In INFOCOM '07, May 2007.
[17] G. Pandurangan, P. Raghavan, and E. Upfal, "Building low-diameter P2P networks," In FOCS '01, pp. 492-499, 2001.
[18] J. Kangasharju, K. Ross, and D. Turner, "Optimizing file availability in peer-to-peer content distribution," In INFOCOM '07.
[19] L. Guo, E. Tan, S. Chen, Z. Xiao, and X. Zhang, "The stretched exponential distribution of internet media access patterns," In PODC '08, pp. 283 - 294.
[20] "AmazingStore log page," http://admin.amazingstore.org/struts/onlineusers.action.