Supplementary File of the TPDS manuscript

by Zhenhua Li, Jie Wu, Junfeng Xie, Tieying Zhang, Guihai Chen, and Yafei Dai
lzh@net.pku.edu.cn, and jiewu@temple.edu

Abstract—This supplementary file contains the supporting materials of the TPDS manuscript — "Stability-Optimal Grouping Strategy of Peer-to-Peer Systems." It improves the solidity and completeness of the TPDS manuscript.

1 Background

In recent years, Peer-to-Peer (P2P) systems have become the base infrastructure of many Internet applications. To explore Internet edge resources, peers (clients) contact each other directly, whereas in conventional networking systems, clients only communicate with servers. P2P systems can be generally categorized into two kinds of architecture: unstructured (e.g., Gnutella [1], KaZaa [2], and eDonkey [3]), and structured (e.g., Chord [4], Pastry [5], Tapestry [6], and Kademlia [7]). The structured architecture is also known as the distributed hash table (DHT).

The unstructured architecture is widely deployed for its simplicity of implementation and loose organization, and thus further adopted by most commercial P2P media streaming systems (e.g., Skype [8], PPLive [9], PPStream [10], and UUSee [11]). However, its flooding-based search method does not scale well as the load on each node grows dramatically with the search radius (often called *TTL*) and the system size. Therefore, DHT is presented as an elegant architecture which seems to possess high scalability and search efficiency. Currently, DHT usually acts as an auxiliary facility of unstructured P2P systems. For example, the Kademlia DHT has been applied in BitTorrent [12] and eDonkey.

2 Existing Work Categorization

Existing works can be roughly classified into the following categories:

1) *GiantOnly.* Besides its broad use in unstructured P2P systems like KaZaa, eDonkey, and Skype, the *GiantOnly* strategy is also employed in some DHT-based deployment schemes, like OpenDHT [13], where only giants can play the role of DHT nodes. Dwarfs are not allowed to enter the DHT, but are instead treated as clients. In essence, GiantOnly compromises scalability for stability, which is reasonable in OpenDHT as OpenDHT is deployed on Planetlab [14], an open platform with sufficient giants.
2) *TotallyFlat.* Despite their substantial difference in overlay organization, Gnutella and Chord both construct a *Totally Flat* world for their participants. That is to say, all peers are equal in function, no matter whether they are giants or dwarfs. Contrary to GiantOnly, TotallyFlat compromises stability for scalability. As any dwarf can provide core services (e.g., data indexing, message routing, and overlay maintenance), TotallyFlat behaves poorly when facing high-churn network environments.
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. [15] focus on the issue of selecting a subset of the available node-set as relatively stable neighbors to replace failed ones. Yeung and Kowk [16] model the neighbor selection process as a cooperative game so that peers form stable coalitions with high possibilities. Nevertheless, the StableNeighbor strategy is more-or-less a zero-sum game, so it is not an essential way to get out of our dilemma. Stable peers do not grow in number or quality, but just shuffle among the neighborhoods of peers.

3 Related Work

We have briefly reviewed related works in Section 2 in three categories: 1) GiantOnly; 2) TotallyFlat; 3) StableNeighbor. Here, we discuss them more and some other issues relevant to the grouping strategy, stability, scalability, etc.

OpenDHT [13] is the representative of the GiantOnly strategy. It runs a DHT on Planetlab nodes [14] to offer services to unstable nodes that play the role of clients. Maintenance cost is greatly reduced since Planetlab nodes are workstations or servers that are always in session, viz., far more stable than typical P2P peers. Wang et al. [17] delve further into this area of study and obtained a threshold to precisely decide how much stability is required for a node to be accepted by the DHT.

For the TotallyFlat strategy, since unstable nodes are unavoidably included, data replication is a commonly used way to enhance data availability. Blake et al. [18] make a comprehensive study on several conflicting system metrics, i.e., stability, scalability, dynamism, and bandwidth consumption. A simple model is proposed, from which they draw the conclusion that it is

invariable to use a large amount of cross-system bandwidth for good stability and scalability in highly dynamic systems. Our work does not intend to enhance system stability by consuming extra communication bandwidth. Instead, we improve system stability by organizing nodes in a better way.

The StableNeighbor strategy investigates how to properly detect and replace failed neighbors with stable neighbors, in order to enhance system stability. Godfrey et al [15] focus on the issue of selecting a relatively stable subset of the available nodes as neighbors. They pay particular attention to a range of different node selection strategies and finally conclude that the simple strategy of picking a uniform-random replacement performs surprisingly well. Henceforth, simply adding some randomization is an easy and effective way to increase stability. Leonard et al. [19] study the churn resilience of various random graphs under various user lifetime models. They examine two metrics: 1) isolation time, and 2) isolation time probability, of the models with and without neighbor recovery, respectively. Similar to [15], a node is deemed to be isolated whenever all of its neighbors have failed. On the contrary, we investigate how long a node or group *itself* survives to describe the system stability. Leonard et al. derive from theoretical analysis that the k-regular graph exhibits the highest level of fault resilience, though maintaining the k-regular property would incur considerable bandwidth consumption.

Hierarchical P2P protocols are invariably the focus of researchers, owing to their practicality and feasibility in our real world. Among them, Gnutella v0.6 [20] used to be the most prevalent one, yet in recent years, it seems to be more attractive to deploy the top-level overlay in a DHT manner (similar to the server overlay in OpenDHT [13]). From a real trace of PPLive, Wang et al. [21] found out that plenty of stable nodes exist (but only in a per-snapshot view). Thus, they propose a tiered overlay design for P2P video streaming, with stable nodes being organized into a tier-1 backbone to serve tier-2 nodes.

The intra-group organization structure deserves further research. Some analytical models have been recently derived to shed light on various structures from various perspectives. For instance, Zoels et al. [22] compare three intra-group structures: 1) single-connection intra-group structure, 2) fully-meshed intra-group structure, and 3) DHT intra-group structure. They draw the conclusion that the first one is superior to the other two in the sense of traffic cost. Our work differs from [22] in that we cluster homogeneous nodes rather than group giants and dwarfs together, so the single-connection structure (which needs a giant in each group) is not applicable to our work. The fully-meshed structure constructs a highly resilient intra-group communication scheme, but would bring too much traffic cost to maintain $O(\frac{1}{2}|G|^2)$ intra-group connections. The DHT structure is too complicated to implement inside a group. Moreover, because most groups are small in size, organizing the few nodes in a group into a DHT is really unworthy. In our paper, we adopt a loosely-meshed structure to organize intra-group nodes. This scheme possesses reasonable resilience, and meanwhile, avoids the enormous traffic cost of the fully-meshed structure and the complication of the DHT structure.

4 Proof of Theorem 1

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

Proof: We complete the proof by reducing the *Minimum Sum of Squares* (MSS) problem to MSG in polynomial time. The MSS problem presented below has proven to be NP-hard [23]:

Definition 1 (Minimum Sum of Squares Problem)**.**
Instance*: A finite set A, the size $s(a) \in Z^+$ for each $a \in A$, and an integer $K \geq 2$.*
Solution*: A partition of A into K disjoint sets $A_1, A_2, \ldots, A_K$.*
Measure *to minimize:* $\sum_{i=1}^{K}(\sum_{a\in A_i} s(a))^2$.

Some trivial differences in terminology are first pointed out: $\mathbb{S}$, m, and G_k in MSG correspond to A, K, and A_i in MSS, respectively. We will reduce MSS to an auxiliary problem MSG' being the same to MSG, except that its measure is $\mathbb{E}(\Psi) = \frac{1}{m}\sum_{k=1}^{m}\psi_k$. Algorithm 1 exhibits how to derive τ_k from $n_i \in G_k$ in polynomial time. Thus, we conclude that the algorithm to deduce ψ_k from G_k can be reduced from $\sum_{a\in A_i} s(a)$ in polynomial time by delicately tuning $s(.)$. Up to now it is possible to reduce MSS to MSG', thus MSG' is NP-hard. Furthermore, $\mathbb{V}ar(\Psi) = \mathbb{E}(\Psi^2) - \mathbb{E}^2(\Psi)$ is more complicated than $\mathbb{E}(\Psi)$, therefore, it is easy to prove by the reduction that MSG is NP-hard. □

5 Proof of Theorem 2

The stability of a group G_k at slot st is

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

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

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

Proof: G_k is empty at the st-th slot if and only if, at the st-th slot, the sessions that started before (called old sessions) have ended and no new nodes join G_k at that time:

$$\begin{aligned}\mathbb{P}(\phi_k(st)) = \mathbb{P}(&\text{no new nodes entered } G_k \text{ at } st)\cdot\\ &\mathbb{P}(\text{all old sessions of } G_k \text{ have ended before } st)\\ &= I(st,k)\cdot II(st,k).\end{aligned} \tag{2}$$

Algorithm 1: Get_τ_k_from_G_k

```
Data: G_k = {n_1, n_2, ..., n_|G_k|}
Result: τ_k
begin
    Put G_k into array a[2|G_k|] in ascending order;
    Suppose a[0] is n_t.joinTime;
    s ⇐ 0;
    pivot ⇐ 0;
    Push a[0] into a stack;
    while pivot < 2|G_k| − 1 do
        for i = 1 to 2|G_k| − 1 do
            if a[i] is n_t.leaveTime then
                s ⇐ s + n_t.leaveTime − n_t.joinTime;
                pivot ⇐ i + 1;
                break;
            if a[i] is a joining time then
                Push;
                else
                    Pop;
    τ_k ⇐ s;
```

Node arrivals are considered independent, so

$$I(st,k)=\sum_{i=0}^{+\infty} v_{st}(i)\Big(1-\big(D(y_k)-D(y_{k-1})\big)\Big)^i. \quad (3)$$

The meaning of $v_{st}(i)$ and $D(y_k)$ can be found in Table 2 of the TPDS manuscript.

The old sessions of G_k are considered to have started at any slot from 0 to $st-1$ independently, and have ended before st. So,

$$II(st,k)=1-\sum_{x=0}^{st-1}\mathbb{P}(\text{at least one session of } G_k \text{ started at } x \wedge \text{its } stl \geq st-x)=1-\sum_{x=0}^{st-1} III(st,k,x). \quad (4)$$

The number of node arrivals at the st-th slot theoretically range from 0 to $+\infty$. Thus,

$$III(st,k,x)=\sum_{i=0}^{+\infty}\mathbb{P}(J_k(i,x)\wedge(S_k(i,st-x))) =\sum_{i=0}^{+\infty}\mathbb{P}(J_k(i,x))\cdot\mathbb{P}(S_k(i,st-x)), \quad (5)$$

where $J_k(i,x)$ is the event that i nodes join G_k at the x-th slot, and $S_k(i,st-x)$ is the event that at least one node out of these i nodes has the $stl \geq st-x$.

Following Bernoulli distribution,

$$\mathbb{P}(J_k(i,x))=\sum_{j=i}^{+\infty} v_x(j)\binom{j}{i}\mathcal{P}^i(1-\mathcal{P})^{j-i}, \quad (6)$$

where $\mathcal{P}=D(y_k)-D(y_{k-1})$.

Due to the independence of each stl,

$$\mathbb{P}(S_k(i,st-x))=1-(1-\mathbb{P}(S_k(1,st-x)))^i, \quad (7)$$

and as G_k's stl belongs to $[y_{k-1},y_k)$,

$$\mathbb{P}(S_k(1,z))=\begin{cases}1 & z<y_{k-1}\\ \frac{D(z)-D(y_{k-1})}{D(y_k)-D(y_{k-1})} & y_{k-1}\leq z\leq y_k\\ 0 & y_k<z\end{cases} \quad (8)$$

Equations (1)-(8) complete the proof. □

6 Grouping Strategy Design

In this section, we describe the design details of our proposed grouping strategy. For either an unstructured system or a DHT system, each group has a single overlay *ID* which is shared among its members. Each group corresponds to one overlay unit (e.g., a Gnutella node or a Chord node). The establishment and maintenance of inter-group links are based on the original overlay specifications. In a group G_k, at any time there is at most one group leader, which acts as a proxy/gate of this group. More specifically, the group leader is responsible for maintaining inter-group links, inter-group message routing, and intra-group coordinating.

Online members of a group are organized into a loosely-meshed structure with only one condition: they must be connected. In fact, there exist at least four methods (i.e., single-connection, fully-meshed, loosely-meshed, and DHT) to organize the intra-group nodes, which has been discussed in Section 3. Each member of group G_k maintains a member list of G_k. When an offline member gets online, its member list will be partially out of date, and it can contact any online member for the latest list. Besides, the member list is critical in checking or enhancing G_k's connectivity.

Since the intra-group nodes are loosely organized, the feasible intra-group routing methods are flooding, random walk, k-random walk, etc. In Section 2.2 of the TPDS manuscript, we have deduced that the average group size is $\frac{N}{m}\leq d^{TTL-TTL'}$ (for unstructured systems) or $\frac{N}{m}\in O(logN)$ (for DHT systems). In either case, the group size is usually small or quite limited. Therefore, we adopt the flooding method for intra-group routing for its simplicity and resilience. Because the group size is relatively small, the message cost of intra-group flooding is in $O(\frac{N}{m})$ and the corresponding flooding radius (TTL) is in fact unnecessary.

6.1 Maintenance

In a group G_k, each node periodically floods its state information (including its inter-group routing table) into the group for intra-group maintenance. So, the members of G_k have a nearly consistent view of the group state. Usually, the online member with the strongest capability (measured in stl, bandwidth, storage, or other metrics) takes the role of group leader, and then notifies the other online members of G_k.

If the function of G_k is to provide bandwidth relay or computing capability, each member only needs to store the same service software. If the function of G_k is to provide data storage, data should be replicated or erasure-coded among the members, and the specific choice is up to the system designer. Besides, group leaders periodically exchange and update their routing table information, just as common nodes in Gnutella or Chord do.

Suppose we group a Gnutella system into m groups in the way in Section 2.2.1 of the TPDS manuscript. For a group G_k, in a maintenance period, each of its online members floods its state information to the others, and the leader of G_k exchanges its routing table information with some other group leaders. Suppose G_k currently contains n_k online members, and the inter-group degree is a constant d_G. Then, the maintenance message cost for G_k in a period is around $n_k \cdot (n_k - 1) + {d_G}^{TTL'}$. The maintenance overhead analysis of Chord grouping is alike.

When a new node n joins in the system S *for the first time*, n contacts the *bootstrap* node (usually a server) to find the member of which group has the most similar *stl* with n. For our homogeneous grouping strategy, this is quite easy because any member in one group can represent the *stl* status of this group. Almost every P2P system contains a bootstrap node, so we do not add extra facilities. If n chooses G_k to join in, G_k's capability and stability would be improved. But, this may be harmful for the overall stability of S since $\mathbb{V}ar(S)$ would be increased. We propose a simple solution to this problem. For a group G_k with the original size s, when $s < |G_k| < 2s$, it is tolerable; when $|G_k| \geq 2s$, G_k is divided into two groups, with each containing s nodes. How to decide the group size s is described in Section 2.2 and Section 2.4.2 of the TPDS manuscript.

It is too difficult to judge whether a node has left the system *forever*. If a node n has left for i days, we decrease its *stl* to $\frac{1}{i}$. So, a permanent leave just means the *stl* falls below a threshold very close to 0. When more than half of G_k's members are considered to have died (i.e., left forever), G_k would not be able to play the role of a stable service group. Then, G_k is merged to G_{k-1} or G_{k+1}.

We group a P2P system when its user number is *relatively stable*. The number of users N fluctuates slightly, that is, the number of *fully new users* mainly offsets the number of *dead users*. At each time slot, many nodes arrive and many nodes leave, but few nodes are newly joining or permanently leaving. This is the reason why the above-mentioned join and leave events scarcely happen, and thus would not significantly degrade the performance of our grouping strategy. Notably, our proposed grouping strategy is not a static design, because it only demands that the system should have a relatively stable user group, rather than stable online users. To our knowledge, most practical systems, including our AmazingStore and CoolFish, possess a relatively stable user group at present.

Even when facing an open P2P system, where the user number is quite unstable (fluctuates sharply), our grouping strategy still works. The grouping manager recollects all nodes' information and then regroups them at intervals. Surely this operation would incur extra overhead as discussed above, but it is the best choice for performance consideration. Refer to Section 10 for simulation results.

6.2 Search

In unstructured P2P systems, a search query is firstly sent from the query initiator (Q) to the group leader (L). On receiving the query, L firstly sends it to the other members of its group, and then forwards it to the neighboring groups (their group leaders) through inter-group links. Likewise, the group leader of each neighboring group forwards the query to its group members and neighboring groups. The inter-group flooding radius is TTL', a small integer. As a result, the message cost for a query is in $O(\frac{N}{m} \cdot d^{TTL'})$=$O(d^{TTL-TTL'} \cdot d^{TTL'})$=$O(d^{TTL})$. The meaning of d and TTL is in Section 2.2.1 of the TPDS manuscript.

In DHT systems, (the index of) any data object is stored in a determinate group. On receiving a query request from one group member, the group leader L just forwards the query to the next group leader as a routing hop in DHT. After $O(\log m)$ hops, the query arrives at its destined group G_d. Finally, the group leader of G_d floods the query to the other members of G_d, and the member with the matched object replies directly to the query initiator Q. As a result, the message cost for a query is in $O(\log m + \frac{N}{m})$. Since $m \in O(\frac{N}{logN})$, $O(\log m + \frac{N}{m})$ = $O(logN)$.

6.3 Discussions on C_k

A group's service capability C_k can be measured from different metrics. In Section 2.1 of the TPDS manuscript, we have defined $Scalability(S) = m \cdot \overline{C} = \sum_{k=1}^{m} C_k$, and C_k is considered as the time-weighted average of G_k's members' capabilities.

CPU/memory is the most important metric in P2P computing systems, like SETI@home [24] and the GPU project [25]. For a group G_k composed of nodes $n_1, n_2, \cdots, n_g$, the average of $C(n_1), C(n_2), \cdots, C(n_g)$ is a proper estimation of C_k, because computing is usually a delay tolerant issue. $C(i)$ denotes the capability of node i.

However, the above estimation method is not suitable for bandwidth. Bandwidth is critical to P2P media streaming systems (like Skype and PPLive), which are sensitive to bandwidth variation. In Fig. 2 of the TPDS manuscript, the bandwidths of the four dwarfs are 500Kbps (n_1), 300Kbps (n_2), 1Mbps (n_3),

and 700Kbps (n_4), respectively. Their time-weighted average bandwidth is 683Kbps. If 683Kbps is taken as C_k, the Skype system G_k may be assigned to undertake more than 600Kbps media streams. Obviously, when n_1 or n_2 is in service, G_k is totally unable to carry a 600Kbps stream, and thus most streams across G_k will degrade greatly in the quality of service or even break off. Consequently, $\min\{C(n_1), C(n_2), \cdots, C(n_g)\}$ is a reasonable estimation of C_k when bandwidth is the metric. Node bandwidth can be approximately assumed to be proportional to its *stl*. This assumption is supported by the common observation in P2P systems: a stable server-style peer usually has high bandwidth, while an unstable client-style peer usually has relatively low bandwidth. *Thus, in this situation, our homogeneous grouping strategy is the most efficient since* $\min\{C(n_1), C(n_2), \cdots, C(n_g)\}$ *is close to their average.*

The estimation method of storage lies between those of CPU/memory and bandwidth. Still using Fig. 2 of the TPDS manuscript as the example, the storage of the four dwarfs are 300GB (n_1), 100GB (n_2), 800GB (n_3), and 700GB (n_4), respectively. Their time-weighted average storage is 533GB. If 533GB is taken as C_k in a P2P storage system [26], [27], it is all right most of the time because a P2P storage system rarely fills in its peers' storage with more than half. Even if n_1 is in service and its remaining storage cannot meet the requirement, it can simply reject the request and inform the requester to find a new group for storage. The storage process is always time consuming, so the users can tolerate a rejection and redirection in most situations.

6.4 Demo of Our Definition of Stability

Our explanation of system stability in Section 2.1 of the TPDS manuscript was brief and abstract, lacking concrete description. Here, we use a demo metaphor which compares our idea to another more familiar scenario, in hopes of better and easier understanding of our definition of stability.

Suppose we want to operate 100 web sites, and we have 500 servers to support these web sites. One server can only support one web site. Suppose each server can only run for 6 hours per day. Then the node stability is $\frac{6}{24} = 0.25$. We do not know when they fail, and they fail independently. Surely, it is impossible to build a perfect system where every web site has high stability. A natural idea (egalitarianism) is to uniformly allocate 500 servers to 100 web sites, with each web site supported by 5 servers. Then each web site is expected to have the same stability: $\psi = 1 - (1 - 0.25)^5 = 0.763$. For a certain web site (says *A*), you may feel this group stability (0.763) is not satisfactory, and then the only way to improve $\psi(A)$ is grabbing a server from another web site (says *B*). Then, $\psi(A)' = 1 - (1 - 0.25)^6 = 0.822$, but $\psi(B)' = 1 - (1 - 0.25)^4 = 0.684$. The gain in $\psi(A)$ is $0.822 - 0.763 = 0.059$, while the loss in $\psi(B)$ is $0.763 - 0.684 = 0.079$. Obviously, as to the whole system, the loss is larger than the gain. If *A* grabs more servers from *B*, the loss will be much larger than the gain. This simple example indicates that the natural idea (egalitarianism) is in fact the best idea. The symbol of egalitarianism is the minimum variance (zero in this example). This is why we define the system stability as inverse proportional to the system variance.

7 Data Sets, Metrics, and Results Summarization

7.1 Generated data set

One time slot is set as one (simulation) second. The number of node arrivals at each second follows the same Poisson distribution:

$$v_t(i) = e^{-\lambda} \cdot \frac{\lambda^i}{i!}, \tag{9}$$

where $\lambda = \frac{N}{Period}$, which means $\frac{N}{Period}$ nodes enter the simulation system per second on average. Refer to Table 2 of the TPDS manuscript for the meanings of N and $Period$.

Previous literatures [28], [29] observe that the *stl* of human-based P2P systems conforms to the heavy-tailed (i.e., Pareto) distribution, yet other systems consisting of non-human devices (e.g., software agents) exhibit exponential distribution behaviors. The latter usually exhibits much more churn than the former. Thus, our simulation uses the second one:

$$D(y) = 1 - e^{-\frac{1}{\eta}y}, \tag{10}$$

in order to simulate a high-churn scenario with $\eta = 100$. That implies the expected *stl* is 100 seconds, and 63.2% of nodes survive less than 100 seconds. We also discard *stl*s being more than $1,000$ seconds ($1 - e^{-1000/100} \approx 0.99995$) and set $Period = 1000$ seconds. The system is simulated to run long enough ($10,000$ seconds) to collect sufficient data. Here "second" is the *simulation second* rather than real time. Our simulation tool for the generated data set is implemented as a discrete time event generator, and an event is generated or triggered per simulation second. For convenience, we can simply take 1000 simulation seconds as 24 hours in real life.

We compare the performances of our grouping strategy (Grouping), the GiantOnly strategy (GiantOnly), and the TotallyFlat strategy (Gnutella and Chord, respectively). Grouping is run in Matlab R2010a as an optimization problem (the *fmincon(.)* function), which determines $y_1, y_2, \ldots, y_m$ to minimize $\mathbb{V}ar(\Psi)$, and nodes are clustered according to the resultant grouping. The optimization error is 10^{-20} and the number of optimization iterations is 3600. The number of nodes N is set to 1000, for N=1000 is in fact the computation limit in our desktop (Intel E4400 CPU@2GHz, 2GB DDR). For DHT systems (GiantOnly, Chord, and Grouping-Chord) we set $m = \frac{N}{logN} = \frac{1000}{\log 1000} \approx 100$;

for unstructured P2P systems (Gnutella and Grouping-Gnutella), we set $d = 4$, $TTL = 5$, and $TTL' = 3$ (they are the common configurations in existing systems). $m \geq \frac{N}{d^{TTL-TTL'}} = \frac{1000}{4^{5-3}} = 62.5$. For the convenience of comparison, for unstructured systems we also set $m = 100$.

GiantOnly works like OpenDHT [13]. It is simulated as follows: When the system is pretty stable, the top m active nodes in terms of predicted remaining *stl* are deemed as giants and serve as the DHT peers. From that moment to the end, whenever a DHT peer ends its session, the node outside the DHT with the largest remaining *stl* is selected into the DHT, in order to keep the DHT size (m) constant.

7.2 AmazingStore trace

AmazingStore [26] is a P2P storage system deployed in the CERNET [30]. It mainly provides user file sharing and user data online backup (the so-called "network disk"). Its overlay is organized by using an enhanced Kademlia DHT. Each AmazingStore user is enforced to preserve 1GB local disk space for system-wide storage, which means every AmazingStore user has the same storage capacity. We traced all the users' join and leave activity over the course of three months, from Jan. 1, 2010 to Mar. 31, 2010. In total, N = 4854 users have been recorded. Their average *stl* is 1.815 hours per day ($\frac{1.815}{24} = 0.075643$).

In the AmazingStore trace, the node arrivals and *stl*s do not accurately follow the Poisson or exponential distributions mentioned above. Instead, they exhibit great "day-night" difference: At night there are always few online nodes. Thus, we use discrete $v_t(i)$ and $D(y)$ values obtained from the trace to simulate the continuous values. For example, if we know $D(y_1)$ and $D(y_2)$, $D(y')$ ($y' \in [y1, y2)$) can be evaluated as the linear interpolation:

$$D(y') = \frac{y' \cdot D(y_2) + y_2 \cdot D(y_1) - y_1 \cdot D(y_2) - y' \cdot D(y_1)}{y_2 - y_1}.$$

Because N = 4854 is too large for the computation from Equation (2) to (8), we carefully examine the $\sum_{i=0}^{+\infty}$ in Equations (3) and (5) and $\sum_{j=i}^{+\infty}$ in Equation (6) to reduce their computation complexity. In Equations (3) and (5), according to our observations of AmazingStore, i is usually 0, 1, or 2. The situation that more than 2 nodes join in a group G_k at a specific time st scarcely happens. Therefore, $\sum_{i=0}^{+\infty}$ is approximated by $\sum_{i=0}^{2}$. Furthermore, $\sum_{j=i}^{+\infty}$ in Equation (6) is approximated by $\sum_{j=i}^{2}$.

Utilizing the above approximations, our grouping strategy divides these 4854 users into $m = \frac{N}{logN} = 396$ groups. Then, we compare the performance of AmazingStore and AmazingStore-grouping (i.e., the new AmazingStore scheme after grouping) by employing the real trace.

7.3 CoolFish trace

CoolFish [31] is mainly deployed in the CSTNet [32] for P2P live/VoD streaming. Different from AmazingStore, CoolFish is an anonymous system, that is, CoolFish users do not need to register an account for use. So, we can only trace the IP address, join and leave time, and the stable bandwidth of a session. Here, "stable" means the session comes into its stable status. Because it is impossible to identify which sessions belong to the same user, we assume each session corresponds to a unique user and restricts the trace period to within 24 hours.

The trace is divided into 9 disjoint sub-traces, corresponding to Apr. 7, 2010 - Apr. 15, 2010, respectively. The number of nodes contained in each sub-trace ranges from 246 to 351. Our grouping strategy is performed on each sub-trace individually.

7.4 Metrics

We evaluate our grouping strategy and the related works mainly from two aspects: stability and scalability.

Churn rate is defined to measure stability. It is the number of dynamic or offline nodes/groups per churn unit. A churn unit is set at 100 seconds for generated data sets, and 600 seconds for AmazingStore/CoolFish traces. For a group, the intra-group node dynamics is not a churn, but the state of the group is changed. We also use *churn ratio* (churn rate / number of nodes or groups) to describe stability more clearly.

We evaluate scalability from two orthogonal perspectives: *search efficiency* and *system storage capacity*. The search efficiency denotes the average number of messages cost in a search. For generated data sets, a node's storage capacity is set proportional to its *stl*, but the proportionality factor is variable. Specifically, for a node n, its storage capacity $C(n) = \alpha \cdot \tau_n$, where $\alpha \in [0.5, 2.0]$. The system storage capacity is the summation of the online nodes'/groups' storage capacities.

Additionally, we further 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). *System stable storage/bandwidth* denotes the summation of the storages/bandwidths provided by *sufficiently stable* (determined according to specific scenarios) nodes/groups. As explained in Section 6.3, for a stable group G_k composed of nodes $n_1, n_2, \cdots, n_g$, $S(G_k) \approx$ the average of $\{S(n_1), S(n_2), \cdots, S(n_g)\}$, while $B(G_k) = \min\{B(n_1), B(n_2), \cdots, B(n_g)\}$. $S()$ means storage and $B()$ means bandwidth.

Furthermore, we measure the *maintenance overhead* of various relevant systems, using the generated data set. Maintenance overhead presents the average number of messages cost in a churn unit for maintaining the

system states (e.g., routing table, grouping information, and so on). The maintenance mechanism of our grouping strategy is described in Section 6.1.

Finally, we evaluate the *load balance* situation of our proposed grouping strategy, using the AmazingStore trace. The detailed evaluation is in the following section.

7.5 Results Summarization

By using our grouping strategy, the performance evaluations in the TPDS manuscript can be summarized in four aspects:

1) *Stability*: Both the churn rate and churn ratio are greatly reduced on the generated data set and two real-world traces.
2) *Scalability*: Both the search efficiency and storage capacity bear a compromise, but the loss is acceptable.
3) *Stable service capacity*: AmazingStore gains much higher stable storage capacity, and CoolFish gains much higher stable bandwidth capacity.
4) *Maintenance overhead*: The maintenance overhead of grouping is less than that of TotallyFlat (Gnutella/Chord), but is slightly more than that of GiantOnly.

8 Load Balance Evaluation

Fig. 1 and Fig. 2 depict the storage load balance situation of AmazingStore and AmazingStore-grouping, respectively. The storage load is sorted in ascending order. Clearly, our grouping strategy greatly improves the load balance situation of AmazingStore. In Fig. 1, most nodes store very little while a minority of nodes store a majority of data. But, in Fig. 2, the storage of most groups lies between 400 MB and 600 MB.

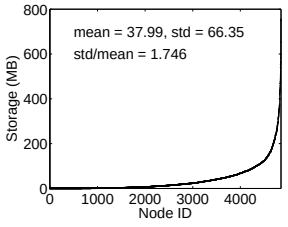

Fig. 1. Load balance of AmazingStore.

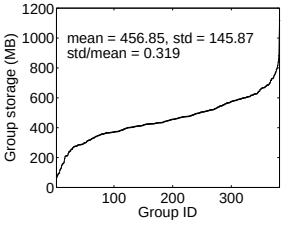

Fig. 2. Load balance of AmazingStore-grouping.

9 Distributed Algorithm

Our proposed grouping strategy in the TPDS manuscript is a centralized mechanism which collects statistic information from all peers, and then designs a global grouping plan. This mechanism brings heavy burden to the central manager, which limits the system scale it can take charge of. A distributed mechanism can overcome this problem, but its performance may degrade to some extent. When the user number N is too large, a proper distributed algorithm for our grouping strategy would be like "divide-and-conquer". All users are divided into several domains according to their affiliated ISPs, ASes (Autonomous System), or some other metrics, and then each domain elects a local server responsible for grouping.

We evaluate the performance of the distributed algorithm using the AmazingStore trace. The centralized algorithm divided $N = 4854$ nodes into $m = 396$ groups. The distributed algorithm uniformly divides all nodes into 5 domains, with each domain containing about 970 nodes. The most stable node in each domain acts like the local server for grouping. From Fig. 3, Fig. 4, and Fig. 5, we discover that both the grouping stability and scalability of the distributed algorithm degrade obviously, compared with those of the centralized algorithm (see Fig. 15, Fig. 16, and Fig. 17 of the TPDS manuscript).

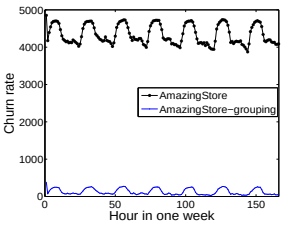

Fig. 3. Churn rates of AmazingStore and AmazingStore-grouping, with distributed algorithm.

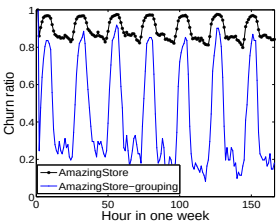

Fig. 4. Churn ratios of AmazingStore and AmazingStore-grouping.

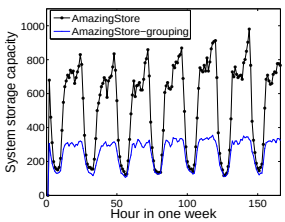

Fig. 5. System storage capacities of AmazingStore and AmazingStore-grouping, with distributed algorithm.

10 Performance In An Open Systems

In an open P2P environment, the session time (stl) of a newly joining node is hard to get although the session time distribution of all nodes can be easily got. Moreover, the total user number (N) may be quite unstable (fluctuate sharply). Then the problem is how to deal with such an open system. In fact, when a new node joins the system, it is impossible to accurately estimate its session time. 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 this new node would be learnt, and then we can allocate it into a more proper group. As to the sharp

fluctuation in user number, our solution is to recollect all nodes' information and then regroup them at intervals. The above solutions would incur extra overhead as discussed in Section 6.1, but it is a simple and correct choice for performance consideration. In this section, we evaluate the corresponding performance through trace-driven simulations.

We still use the AmazingStore trace as described in Section 7.2. This trace involves 4854 nodes in three months. To simulate an open P2P system, assume the open-AmazingStore system scales from the initial state composed of 1000 nodes. In each subsequent week, 300 new nodes join the system. Because we know nothing about the newly joining nodes (no prediction), we estimate their session time as the average session time of existing nodes, and use these inaccurate information to regroup the system. After a new node joins, its session time is gradually learnt from its session history.

We compare the performance of our proposed grouping strategy under different situations: 1) no prediction of node stl, and 2) with prediction of node stl. The results are illustrated in Fig. 6 and Fig. 7. Without prediction of new nodes' stl, our grouping strategy performs slightly worse in $\overline{\Psi}$ and $\mathbb{V}ar(\Psi)$, but the performance degradation is mainly acceptable.

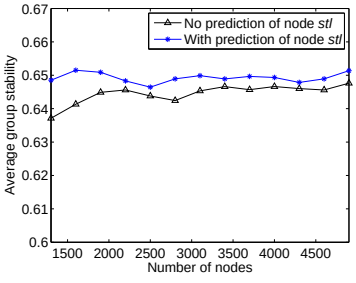

Fig. 6. Evolution of average group stability ($\overline{\Psi}$) as new nodes join in an open P2P system.

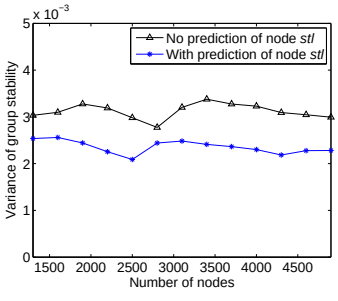

Fig. 7. Evolution of the variance of group stability ($\mathbb{V}ar(\Psi)$) as new nodes join in an open P2P system.

ACKNOWLEDGMENT

We would like to thank Feng Xiao and Song Ding for their help in collecting the AmazingStore trace, Prof. Haiyan Cai for his insightful advice, and Prof. Edward Chan for his comprehensive help.

This work is supported by the National Basic Research Program of China under Grant No. 2011CB302305, and the China NSF grants under Grant No. 60873051, 60825205, and 61073152.

REFERENCES

[1] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the Gnutella Network," IEEE Internet Computing, pp. 50-57, Jan. 2002.
[2] N. Leibowitz, M. Ripeanu, and A. Wierzbicki, "Deconstructing the Kazaa Network," The 3rd IEEE Workshop on Internet Applications, pp. 112-120, Jun. 2003.
[3] K. Tutschku, "A Measurement-based Traffic Profile of the eDonkey Filesharing Service," Lecture notes in computer science, pp. 12-21, 2004, Springer.
[4] 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.
[5] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," IFIP/ACM Conference on Distributed Systems Platforms (Middleware '01), vol. 11, pp. 329-350, 2001.
[6] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," IEEE Journal on selected areas in communications, vol. 22, no. 1, pp. 41-53, 2004.
[7] P. Maymounkov, and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," In IPTPS '02.
[8] S.A. Baset, and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," In INFOCOM '06.
[9] Y. Huang, "Experiences with PPLive," Keynote at ACM SIGCOMM P2P-TV workshop, 2007.
[10] "PPStream web site," http://www.ppstream.com.
[11] "UUSee web site," http://www.uusee.com.
[12] B. Cohen, "Incentives build robustness in BitTorrent," Workshop on Economics of Peer-to-Peer systems, vol. 6, 2003.
[13] 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.
[14] "Planetlab website," http://www.planet-lab.org.
[15] P. Godfrey, S. Shenker, and I. Stoica, "Minimizing churn in distributed systems," In SIGCOMM '06, pp. 147-158, Sep. 2006.
[16] M. Yeung, and Y. Kwok, "Game Theoretic Peer Selection for Resilient Peer-to-Peer Media Streaming Systems," In ICDCS '08.
[17] C. Wang and K. Harfoush, "On the stability-scalability tradeoff of DHT deployment," In INFOCOM '07, May 2007.
[18] C. Blake and R. Rodrigues, "High availability, scalable storage, dynamic peer networks: pick two," In HotOS '03, pp. 18-21.
[19] D. Leonard, Z. Yao, V. Rai, and D. Loguinov, "On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks," IEEE/ACM Transactions on Networking, vol. 15, no. 3, pp. 644-656, 2007.
[20] P. Kirk, "RFC-Gnutella 0.6 Specification," http://rfc-gnutella.sourceforge.net.
[21] 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.
[22] S. Zoels, Z. Despotovic, and W. Kellerer, "On hierarchical DHT systems – An analytical approach for optimal designs," Computer Communications, vol. 31, no. 3, pp. 576-590, 2008.
[23] G. Ausiello, "Complexity and Approximation: Combinatorial Optimiza-tion Problems and Their Approximability Properties," Springer, 1999.
[24] "SETI@home website," http://setiathome.berkeley.edu.
[25] "The Gnutella Processing Unit," http://gpu.sourceforge.net.
[26] "AmazingStore web site," http://www.amazingstore.org/.
[27] Y. Sun, F.-M. Liu, B. Li, B. Li, and X.-Y. Zhang, "FS2You: Peer-Assisted Semi-Persistent Online Storage at a Large Scale," In INFOCOM '09, Rio de Janeiro, Brazil, April 2009.
[28] 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.
[29] 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.
[30] "CERNET web site," http://www.cernet.edu.cn/.

[31] "CoolFish web site," http://www.cool-fish.org/.
[32] "CSTNet web site," http://www.cstnet.net.cn/.