2009 International Conference on Parallel Processing

On Maximum Stability with Enhanced Scalability in High-Churn DHT Deployment

Junfeng Xie, Zhenhua Li, Guihai Chen
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, 210093, P. R. China
Email: {jfx, lizhenhua}@dislab.nju.edu.cn, gchen@nju.edu.cn

Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton Florida, 33431, USA
Email: jie@cse.fau.edu

***Abstract*—When applied in a commercial deployment, DHT-based P2P protocols face a dilemma: although most real-world participants are so unstable that the maintenance overhead is prohibitively high, they must be effectively utilized due to the lack of stable participants. Thus, determining how to leverage unstable nodes to enhance system scalability and then maximize stability in high-churn scenarios becomes a substantial problem. This paper focuses on this topic, and our main findings are two folds: 1) we propose a homogeneous grouping scheme for scalability enhancement. Besides extending system storage capacity by admitting all nodes, it clusters homogeneous nodes together, deploys the inter- and intra-group connections distinctively, and tunes the number of groups, which aims to facilitate search efficiency; 2) we further look into how to maximize stability under this scheme, which is formulated as the problem *Maximum Stability of Grouping*. It not only proves to be NP-hard, but also infeasible; therefore, we propose an approximated grouping approach and reduce it to an optimization problem that proves to be feasible. Simulation results exhibit that our grouping strategy effectively captures the stability-scalability tradeoff. Based on our proposed measurement metrics, it doubles the storage capacity of so-called GiantOnly strategy by incurring slightly more churn and search latency, and is about four times as stable as Chord with equal capacity and mild improvement in search efficiency.**

***Index Terms*—Peer-to-peer, distributed hash table, stability, scalability, grouping, homogeneity, high churn, optimization**

I. Introduction

In recent years, peer-to-peer (P2P) systems have been widely deployed as the base infrastructures of many Internet applications. Three of their prominent applications include BitTorrent [1], Skype [2] and eDonkey [3]. To explore Internet edge resources, peers (clients) contact each other directly, whereas in conventional networking systems, clients only communicate with its server. P2P systems are categorized into three kinds of architecture: centralized, decentralized and unstructured, and DHT (distributed hash table)-based. Scalability and efficient search enable DHT-based protocols to be more applicable for large deployment, thereby attracting more attention from networking communities. Among them, Chord [4] is selected as the prototype of our work. In an N-node network, Chord locates arbitrary data in $O(\log N)$ hops.

Previous studies [5]–[7] show that real-world P2P systems have to involve many unstable participants (e.g., PCs and PDAs with modems). We call them *dwarfs* due to their small *session time length* (***stl***). Their participation invariably induces high churn. To reduce the consequential maintenance cost for data movement and routing table renewal [8], some DHT-based deployment schemes (e.g., OpenDHT [9]) only employ nodes with large ***stl***s, e.g., servers or workstations that are always online (correspondingly, they are called *giants*). Dwarfs are not allowed to enter the DHT, but are instead treated as clients. We coin the term *GiantOnly* to refer to this strategy. GiantOnly essentially compromises scalability for stability, which is reasonable in OpenDHT as OpenDHT is deployed on Planetlab [10], an open platform with sufficient giants.

Nevertheless, some other large-scale commercial deployment schemes (e.g., Skype [2], KaZaA [11] and eDonkey [3]) are mainly deployed on Internet edges where dwarfs constitute the dominant proportion. Should GiantOnly be opted for in such scenario, it means more service requests are imposed on fewer service providers, which seriously hinders system scalability.

The motivation of this paper is based on the observation that in high-churn DHT deployment, the capability of a single dwarf is negligible, but due to their overwhelming proportion, they are still able to make significant contributions with their combined efforts. More importantly, the exploration of dwarf capability seems to be the only choice when attempting to offer scalable services in some high-churn scenarios with few giants. A sophisticated organization protocol is needed to mitigate the consequential deterioration in stability.

In summation, the major question becomes: *How can we leverage dwarfs in order to enhance system scalability and then maximize system stability in high-churn scenarios?*

Our primary idea is to admit all nodes and cluster homogeneous nodes together (i.e., dwarf with dwarf, giant with giant; we defer explaining the reason until Section III-A). The inter- and intra-group connections are deployed distinctively, and the number of groups is tuned to ease search latency.

We categorize each group into one of two modes - offline or online - at some slot. Due to the member cooperation in file storage (addressed in Section III-A at length), a group is online only if at least *one* member is active (i.e., in session) at that time; otherwise it is offline. Correspondingly, a group's *survival time length* during some period is defined as the total number of slots (within that period) when it is online.

Here we temporarily define a group's stability as its survival

0190-3918/09 $26.00 © 2009 IEEE
DOI 10.1109/ICPP.2009.12

time length. Our grouping strategy makes the improvement in group stability like 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 equalize the stability levels across all groups to maximize the overall stability from a system perspective. Based on the definition of group stability, this equalization is achieved by differentiating each group size.

The example depicted in Table I illustrates this idea. We determine that by assigning more nodes to a dwarf group and less to a giant one, the dwarf group can survive for a time period equal to that of its giant counterpart. Such a discriminative grouping strategy ought to apply to a more comprehensive definition of group stability, as well.

To sum up, our contributions are enumerated as follows:

1) We propose a homogeneous grouping scheme to improve scalability. Generally speaking, apart from admitting all nodes to extend system storage capacity, it clusters homogeneous nodes together, deploys the inter- and intra-group connections distinctively, and tunes the number of groups to alleviate search latency;
2) Under this scheme, we formalize the problem *Maximum Stability of Grouping* (MSG) to explore a particular grouping approach for stability maximization. It is both NP-hard and infeasible, hence we propose an approximated grouping approach and prove that it can be reduced to a feasible optimization problem.

A simulation is implemented to compare our Grouping with GiantOnly and Chord. The results illustrate that Grouping captures a more reasonable tradeoff between stability and scalability in high-churn scenarios: based on our measurement metrics, it gains storage capacity one time more than *GiantOnly* by mildly jeopardizing stability and search efficiency, and is around four times as stable as Chord with the same capacity and even less search latency.

The rest of the paper is organized as follows. After surveying related research in Section II, we introduce in Section III the homogenous grouping scheme, formulate MSG, prove its NP-hardness, and elaborate on its infeasibility. We further give the approximated grouping approach and reduce it into a feasible optimization problem. The simulation is implemented and its results are analyzed in Section IV. Section V concludes the paper and discusses some future work.

TABLE I
GIANT GROUP VERSUS 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}												●	●	●		

Notice: 1) The black dot for node n_i at the j-th slot indicates that n_i is active at that slot, e.g., n_6 joins at the 9-th slot and leaves at the 11-th slot; its ***stl*** is 3 slots. 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 after the 5-th slot. Owing to such relay, at least one active node throughout. In accordance with our definition of group stability, the dwarf group is stable during this period.

II. Related Work

Stability is intensely studied in the research community due to the inherent high-churn of P2P systems. Previous work is classified into two categories: 1) choosing appropriate stable nodes as DHT members, and 2) investigating neighbor failure detection and recovery mechanisms.

OpenDHT [9] runs a DHT on Planetlab nodes [10]. It offers services to unstable nodes that play the roles of clients. The 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. [12] delve further into this line of study and obtain a threshold to precisely decide how much stability is required for a node to be accepted by the DHT. Similarly, from a real trace of PPLive, Wang et al. [13] find out that plenty of stable nodes exist (but only in a per-snapshot view). They thus strive to leverage them to serve other unstable nodes specifically from the aspect of video streaming.

Other work investigates how to properly detect and replace failed neighbors in order to enhance system stability. Godfrey et al. [6] focus on the issue of selecting a subset of the available node set as neighbors to replace failed ones. 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 reduce churn. Leonard et al. [14] model the k-regular graph and examine two metrics - isolation time and isolation time probability - for models with and without neighbor recovery, respectively. Similar to [6], 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 system connectivity. Leonard et al. derive from theoretical analysis that the k-regular graph exhibits the highest level of fault resilience. Like [6], they investigate the effect of the neighbor recovery mechanism and prove that system stability is significantly improved, though the consequential bandwidth consumption is non-negligible.

When unstable nodes are unavoidably included, data replication is an alternative way to enhance data availability. Blake et al. [8] give 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

TABLE II
TERMINOLOGY

Notation	Definition
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.
τ_k	G_k's survival time length.
σ_k	G_k's access rate.
B_X/B_x	B_X is the random variable of node capability. B_x is its sampling, i.e., B_x is the capability value of the node whose ***stl*** equals x.
C_k	G_k's capability.
$D(y)$	CDF of ***stl*** (y).
F_k	the number of files in G_k.
G_k	the k-th group.
m	the number of groups.
N	estimated network size.
$v_t(i)$	PDF of the number of arrivals (i) at the t-th slot.
st	a stable slot, viz., the slot when the system size has been stable.

scalability in highly dynamic systems. Contrariwise, our work does not intend to consume bandwidth for system stability, but rather organizes nodes to reduce such costs.

Hierarchical P2P protocols are invariably the focus of researchers owing to its practicality and feasibility for real world. Among them, Gnutella 0.6 [15] 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 [9]). Some analytical models have been recently derived to shed light on these structures from various perspectives. For instance, Zoels et al. [16] strive to minimize the traffic cost for lookup, maintenance and republish. They compare three hierarchical systems, viz., single-connection intra-group structure, fully-meshed intra-group structure and DHT intra-group structure and reach the conclusion that the first one is superior to the other two in the sense of traffic cost. Our work differs from [16] in that we cluster homogeneous nodes (rather than group giants and dwarfs together) in an effort to maximize stability.

III. GROUPING STRATEGY

In this section, we first describe the homogeneous grouping scheme - the general principles to organize nodes in a scalable way. Furthermore, we analyze how to maximize stability under this scheme by formulating the problem Maximum Stability of Grouping. It proves to be NP-hard, and is also infeasible, thereby we propose an approximated grouping approach by introducing some plausible constraints and realistic assumptions. This approach is reduced to a feasible optimization problem. The grouping scheme and approach together compose our grouping strategy.

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

A. *Homogeneous Grouping Scheme*

We design the grouping scheme so as to enhance system scalability from two orthogonal perspectives, namely, *system storage capacity* and *search efficiency*.

To extend storage capacity, the system admits all nodes (rather than only giants as in GiantOnly) and assigns them into different groups. Previous studies (e.g., [5] and [17]) have observed that nodes with long session time are usually servers or workstations with a dual ISDN or cable, i.e., they are also giants in other aspects (bandwidth, storage, etc), and vice versa. It is therefore acceptable to regard a node's ***stl*** proportional to its general capability in further analysis.

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 that clusters giants and dwarfs in a mixed way is greatly invalidated. Such idea will also induce low efficiency when taking the dramatic asymmetry in node capability into account. For instance, bandwidth asymmetry prevents giant bandwidth from being fully-leveraged when it communicates with dwarfs, and because of the huge diversity in storage, the departure of a giant may make it hard for the remaining dwarfs to take over its many files. Furthermore, the users (nodes) of each ISP usually exhibit certain homogeneity, so the homogeneous grouping scheme has the potential to facilitate topology awareness, as well. In a nutshell, it is both reasonable and efficient, at least in high-churn scenarios, to group nodes homogeneous in terms of capability (equivalently, in terms of ***stl***).

To ease search latency, a hierarchical structure is employed for system connection and routing. Each group corresponds to one DHT unit (e.g., a *virtual* Chord node). The establishment and maintenance of inter-group connections are based on the DHT, and are implemented as follows: if G_a requires a DHT-connection to G_b, multiple links from G_a members to G_b members are established for resilience. In contrast, the intra-group connections are random. The members of each group hold the same DHT ID, maintain *the same* (from the perspective of the DHT) routing tables, cooperate to store files, etc. File storage is based on erasure coding [18] such that users are able to get the complete file from only a fraction of all its fragments. Two benefits follow: for the graceful departure of a node, it is unnecessary to transfer all of its fragments to other active members, which conserves bandwidth. Moreover, the system becomes more resilient to unexpected node crashes. A search consists of two successive phases: 1) the DHT-based route among intermediate groups to arrive at the target DHT unit (called destination group) where the desired file should locate, and 2) the unstructured route within that destination group to integrate enough file fragments. Figure 1 exemplifies the structure.

Although our target scenarios are highly dynamic, many previous literatures (such as [5], [17]) have observed that the number of departures will be roughly offset by those of arrivals over time, i.e., the network size is eventually stable. For this reason, the system size is supposed to be estimated from historical records up to some constant factor and we denote it as N.

In the case where N is known, it is perceivable that, with a small number of groups m - which implies big group sizes -

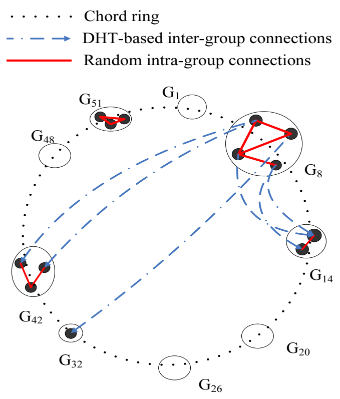
Chord ring
DHT-based inter-group connections
Random intra-group connections
G51
G1
G48
G8
G14
G42
G32
G26
G20

Fig. 1. An example of the hierarchical structure based on Chord. The ID space is $[0, 64)$. The nodes are supposed to be clustered into nine groups, and only $G_8, G_{14}, G_{32}, G_{42}$ and G_{51} are online at that time. The inter-group connections are based on Chord, e.g., the fingers of G_8 ought to point to G_{14}, G_{32} and G_{42} (multiple links for resilience). The fingers of other groups are omitted for brevity.

the system is unscalable, in that the unstructured route within a large group is prohibitively time-consuming. On the contrary, the system ought to be stable as the group with a bigger size is more likely to survive. It is Gnutella-like for one group to contain all system nodes. Oppositely, a large m leads to better scalability but worse stability. It is Chord-like for each group to be comprised of only one node. In a word, m needs to be delicately tuned to alleviate search latency. In the following theorem we will briefly prove that $m \in \Theta(N/\log N)$ turns out to be a plausible choice.

Theorem 1. *Suppose $m \in \Theta(N/\log N)$ (that is to say, the average group size N/m is in $\Theta(\log N)$); the number of search hops is in $O(\log N)$ on average.*

Proof: The number of search hops is in $O(\log m) = O(\log(N/\log N))$ for the DHT-based route. That number involved in the unstructured route depends on several factors, such as 1) the particular topology of its destination group, 2) the search mechanism deployed, e.g., random walk [19] (from 1-walker to k-walkers) and flooding, and 3) the parameter settings in erasure coding, etc. Nevertheless, it is perceivable that even in the worst case, viz., 1-walker conducted in any graph without erasure coding, that number's expectation should still be less than the average size of all groups, i.e., in $O(\log N)$. The total number is thus in $O(\log(N/\log N)) + O(\log N) = O(\log N)$. ■

Henceforth, $m \in \Theta(N/\log N)$ renders an acceptable search latency's upper bound - $\log N$, though besides the factors aforementioned, the coefficient still depends on the variance of group sizes and access rate.

B. Maximum Stability of Grouping (MSG)

The homogeneous grouping scheme merely imposes two *general* restrictions (hence called scheme), namely, homogeneity and $m \in \Theta(N/\log N)$, so as to enhance scalability. It is thus possible to further design a *particular* approach to group nodes under this scheme in an effort to maximize stability. We will formalize this problem as MSG after defining relevant metrics.

Recall the analogy between the grouping strategy and a zero-sum game mentioned in Section I. Each group's stability ought to be equalized to maximize overall system stability under the homogeneous grouping scheme. Following this insight, system stability can be measured by the variance of group stability (a random variable denoted as Ψ), and estimated by an unbiased estimator:

$$Var(\Psi) = \frac{1}{m-1}\sum_{k=1}^{m}(\psi_k - \overline{\psi})^2, \tag{1}$$

where ψ_k is G_k's stability, $\overline{\psi} = \frac{1}{m}\sum_{k=1}^{m}\psi_k$, and stability maximization is equivalent to $Var(\Psi)$ minimization.

It is insufficient to simply define ψ_k as G_k's survival time length (denoted as τ_k) like in Table I. This is because groups (nodes) have files with different popularity and are thus not accessed with identical frequency [20]. Taking this phenomenon into consideration, we define ψ_k as τ_k weighted by G_k's access rate σ_k:

$$\psi_k \triangleq \frac{\tau_k}{\sigma_k}, \tag{2}$$

and σ_k is defined as the popularity summation of all G_k's files:

$$\sigma_k \triangleq F_k \cdot f(G_k), \tag{3}$$

where $f()$ is some kind of empirical formula to predict the average popularity of G_k's files and F_k is the total number of G_k's files.

σ_k's definition implies that in a stable system, the more popular files a group has, the longer it survives. All ratios between ***stl*** and access rate will be about identical in the most stabilized system where $Var(\Psi)$ is minimized.

We denote the set of nodes that will join the system during a significant period by $S = \{n_1, n_2, \ldots, n_L\}$. Assume each node n_i's access rate $n_i.ar$, joining time $n_i.joinTime$ and leaving time $n_i.leaveTime$ are *a priori* knowledge. Without the homogeneity restriction taken into account, MSG is formalized as follows:

Definition 1 (Maximum Stability of Grouping)**.**
Instance: $S = \{n_1, n_2, \ldots, n_L\}$.
Solution: A partition of S into m disjoint sets $G_1, G_2, \ldots, G_m$.
Measure: $Var(\Psi) = \frac{1}{m-1}\sum_{k=1}^{m}(\psi_k - \overline{\psi_m})^2$.

C. NP-hardness

Theorem 2. *With a non-trivial $m \geq 1$, MSG is NP-hard.*

Proof: Minimum Sum of Squares (MSS) presented below has proven to be NP-hard [21]:

Definition 2 (Minimum Sum of Squares)**.**

Instance: Finite set A, 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: $\sum_{i=1}^{K}(\sum_{a\in A_i} s(a))^2$.

We complete the proof by reducing MSS to MSG in polynomial time. Some trivial differences in terminology are first pointed out: S, m and G_k in MSG corresponds 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 in the appendix exhibits how to derive τ_k from $n_i \in G_k$ in polynomial time. Following Equation (3), σ_k is a simple function of G_k. Combined with Equation (2), 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, $Var(\Psi) = \mathbb{E}(\Psi^2) - \mathbb{E}^2(\Psi)$ is more complicated than $\mathbb{E}(\Psi)$, it is thereby easy to prove by the reduction to absurdity that MSG is NP-hard whenever MSG' is. ■

Homogeneity restrictions can be further imposed on the problem, yet this exerts no influence on its NP-hardness.

D. Approximated Grouping Approach

Besides intractability, MSG is also infeasible in that it entails the prediction of *each* node's joining and leaving time, unpractical in real systems. We thereby look into this issue from another perspective. Our approach deploys homogeneity more restrictively so as to reduce MSG into an optimization problem where only the distribution of ***stl*** - $D(.)$, the number of arrivals - $v.(.)$ and the empirical formula of file popularity - $f(.)$ need to be known. To this end, 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. This seems to somehow jeopardize system stability by prohibiting any overlap of different ***stl*** ranges, but this should be insignificant.

Theorems 3 and 4 give the proof that the approach is actually an optimization problem with only $f(.)$, $D(.)$ and $v.(.)$ as known conditions.

Theorem 3. *σ_k is the function of y_{k-1} and y_k.*

Proof: Equation (3) implies that F_k needs to be calculated to obtain σ_k. Numerous literature [22] address the load balancing problem and have proposed various mechanisms. This paper focuses only on the high-churn scenarios so it is simply assumed that files can be assigned to G_k proportional to its capability C_k with the use of existing load balancing mechanisms. Without loss of generality, we regard them to be numerically equal:

$$F_k = C_k. \tag{4}$$

As G_k members have different capabilities, we approximate C_k with their average:

$$C_k = \mathbb{E}_k(B_X), \tag{5}$$

where B_X is the random variable of node capability (X is the random variable of ***stl***), and $\mathbb{E}_k$ implies that only G_k's nodes are involved in the operation.

Node ***stl*** x is assumed to be proportional to its capability B_x, thereby X and B_X can be regarded equal in distribution, viz., $B_X \sim D(X)$ ($D(X)$ is CDF of X), so

$$\mathbb{E}_k(B_X) = \frac{1}{D(y_k) - D(y_{k-1})}\int_{y_{k-1}}^{y_k} x dD(x). \tag{6}$$

Equations (3)-(6) complete the proof. ■

To simplify the analysis, we sample a slot st large enough such that the system size has been relatively stable (e.g., it slightly fluctuates around an estimated value) at that time. We re-define τ_k as the probability that G_k is online at the st-th slot:

$$\tau_k \triangleq 1 - \mathbb{P}(\phi_k(st)), \tag{7}$$

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

Theorem 4. *τ_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 or *old* for brevity) have ended and no new nodes join G_k at that time:

$$\begin{aligned}\mathbb{P}(\phi_k(st)) &= \mathbb{P}(\text{no new at } st)\cdot\mathbb{P}(\text{old ended before } st)\\ &\triangleq I(st,k)\cdot II(st,k).\end{aligned} \tag{8}$$

Node arrivals are independent, so

$$I(st,k) = \sum_{i=0}^{+\infty} v_{st}(i)\Big(1 - \big(D(y_k) - D(y_{k-1})\big)\Big)^i. \tag{9}$$

The old sessions started at any slot from 0 to $st-1$ independently, and end before st is equivalent to the shortness of their ***stl***s, i.e.,

$$\begin{aligned}II(st,k) &= \prod_{x=0}^{st-1}\mathbb{P}((\text{old started at } x)\wedge(\textbf{\textit{stl}} < st - x))\\ &\triangleq \prod_{x=0}^{st-1} III(st,k,x).\end{aligned} \tag{10}$$

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

$$\begin{aligned}III(st,k,x) &= \sum_{i=0}^{+\infty}\mathbb{P}(J_k(i,x)\wedge(S_k(i,st-x)))\\ &= \sum_{i=0}^{+\infty}\mathbb{P}(J_k(i,x))\cdot\mathbb{P}(S_k(i,st-x)),\end{aligned} \tag{11}$$

where $J_k(i, x)$ is the event that i nodes join G_k at the x-th slot, and $S_k(i, z)$ is the event that the ***stl***s of i nodes in G_k are all less than z slots.

Following Bernoulli distribution,

$$\mathbb{P}(J_k(i, x)) = \sum_{j=i}^{+\infty} v_x(j) \binom{j}{i} \mathcal{P}^i (1 - \mathcal{P})^{j-i}, \qquad (12)$$

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

Due to the independence of each ***stl***,

$$\mathbb{P}(S_k(i, st - x)) = (\mathbb{P}(S_k(1, st - x)))^i, \qquad (13)$$

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

$$\mathbb{P}(S_k(1, z)) = \begin{cases} 0 & z < y_{k-1} \\ \frac{D(z) - D(y_{k-1})}{D(y_k) - D(y_{k-1})} & y_{k-1} \leq z \leq y_k \\ 1 & y_k < z \end{cases} \qquad (14)$$

By Equations (7)-(14), we complete the proof. ■

Following Equations (1), (2) and the above theorems, it is easy to obtain Corollary 1:

Corollary 1. *Our approximated grouping approach can be reduced to an optimization problem where $y_1, y_2, \ldots, y_{m-1}$ needs to be determined to minimize $Var(\Psi)$ and the known conditions are $f(.)$, $D(.)$ and $v.(.)$.* ■

We deem this approximated grouping approach to be feasible in that besides N, $f(.)$, $D(.)$ and $v.(.)$ are admittedly predictable, as well: it is widely assumed in literature that node arrival is a memoryless and stochastic process, hence at the t-th slot, its number conforms to Poisson distribution [23]. Additionally, it is confirmed in [12] and [20] that the distribution of node ***stl*** and file popularity are predictable by monitoring node session history and file retrieval record, respectively.

IV. Performance Evaluation

In this section, we design the simulation with nodes constantly joining and leaving the system. This dynamic process enables us to investigate the stability and scalability of Grouping by contrasting it with Chord and GiantOnly.

A. Simulation Environment Setup

First, file popularity is not considered for the sake of simplicity, so $\psi_k = \tau_k$. Second, two nodes enter the simulation system per slot on average, and one slot is set as one second. The number of node arrivals at each second follows the same Poisson distribution:

$$v_t(i) = e^{-\lambda} \cdot \frac{\lambda^i}{i!}, \qquad (15)$$

where $\lambda = 2$. Third, previous literatures [5], [7] observe that the ***stl*** of human-based P2P systems conform 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 simulation uses the second one:

$$D(y) = 1 - e^{-\frac{1}{\eta} y}, \qquad (16)$$

where to simulate a high-churn scenario, $\eta = 100$. 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 ($1 - e^{-1000/100} \approx 0.99995$). The system is simulated to run long enough ($10,000$ seconds) to collect sufficient data.

B. Measurement Metrics

Churn rate is defined to measure stability. It is the number of dynamic peers per churn unit (set as 100 seconds). A peer represents a group in Grouping and a node otherwise, and it is dynamic if it changes its status (i.e., from online to offline or vice versa) within that unit. We say strategy A is x times as stable as strategy B when B's churn rate is on average x times as much as A.

We evaluate scalability from two orthogonal perspectives - system storage capacity and search efficiency. Recall the assumption that B_X and X are equal in distribution, a node's capacity can be numerically represented by its ***stl*** without loss of generality. To measure search efficiency, particular attention is paid to the number of hops involved in Grouping's unstructured search. Notice that such number is definitely trivial for those small-sized groups (those containing less than 10 nodes in our simulation), thereby it is merely necessary to look into the cases that the eight biggest groups (viz., $G_1, G_2, \ldots, G_8$) act as destination groups.

C. Implementation, Results, and Analysis

We simulate GiantOnly as follows: at the st-th slot when system size is pretty stable, the top m active nodes in terms of predicted remaining ***stl*** are deemed as giants and thus 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, which maintains the DHT size constant (m). Grouping, Chord and GiantOnly are compared from the st-th slot to the end.

For implementing Grouping, it is required to determine the value of st and m. To this end, we plot the evolution of system size - $N(t)$ over time t. Figure 2 is the result after ten iterations. We observe that $N(t)$ is approximately 200 over significant periods of time after the 400-th second. This is consistent with Little's law [24], which states that $N(t)$ converges to the product of the expectations of $v_t(.)$ and $D(.)$, i.e., $\lambda \cdot \eta = 200$. Therefore, $st = 500$, $N = 200$, $m = 200/\log_2 200 \approx 26$, and for comparison purpose, the size of Chord is also set to 200.

The approximated grouping approach is run in Matlab 7.5.0 as an optimization problem which determines $y_1, y_2, \ldots, y_{25}$ to minimize $Var(\Psi)$, and nodes will be clustered according to the resultant grouping. We set the optimization error as 10^{-8} and display the output in Figure 3 and Figure 4. In accordance with their ***stl*** intervals, all groups are sorted in ascending order and indexed accordingly. Just as predicted, 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.

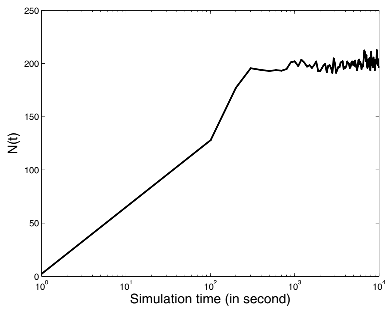
Fig. 2. The evolution of network size over time. The expectations of #node arrivals is 2 and that of ***stl*** is 100. The horizontal axis is log plot.

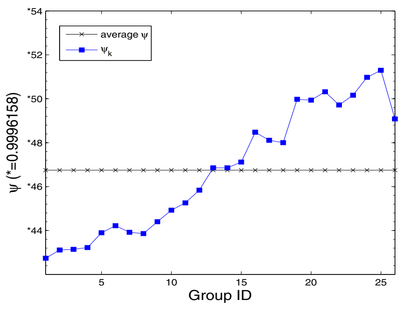
Fig. 3. Each group's stability ψ_k and their mean. The optimization problem is run in Matlab 7.5.0 and the optimization error is set to 10^{-8}.

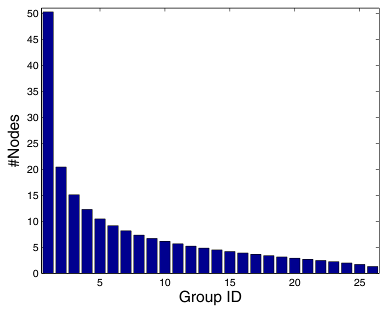
Fig. 4. The number of nodes in each group to gain the stability balance exhibited in Figure 3. Groups are sorted according to their ***stl*** intervals.

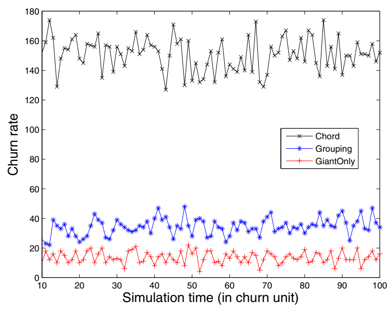
Fig. 5. The evolution of churn rate over time in Grouping, GiantOnly and Chord. One churn unit is set to 100 seconds.

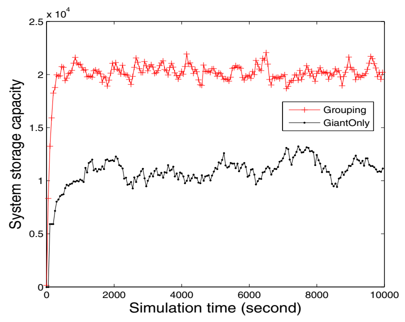
Fig. 6. The evolution of system capacity over time in Grouping and GiantOnly. That of Chord is identical to Grouping therefore omitted.

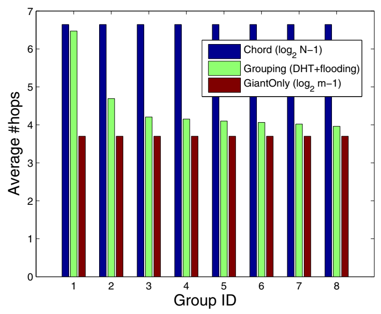
Fig. 7. The average #hops of eight kinds of searches whose destination groups are $G_1, G_2, \ldots, G_8$, respectively.

By consulting the output, each newly-joining node is destined to enter a certain group based on its ***stl***.

We contrast Grouping with the other two strategies in terms of the three metrics. Figure 5 demonstrates that, as we expected, Chord is far more dynamic than Grouping. In most churn units, there exist more than 160 dynamic nodes. On the contrary, GiantOnly's churn rate is low: the number of dynamic nodes are mostly less than 20 per churn unit. In grouping, a group is not likely to be empty owing to its multiple members, thus its churn rate resides between Chord and GiantOnly. On average, Chord's churn rate is 3.1 times more so than that of Grouping. That is to say, Grouping is 4.1 times as stable as Chord. On the other hand, Grouping's churn rate is less than twice as much as that of GiantOnly, which seems pretty stable.

Since all nodes are involved in Chord and Grouping, their storage capacities are approximately equal. Henceforth, we only contrast Grouping with GiantOnly on this metric. Figure 6 explicates that Grouping's capacity is perceivably greater than that of GiantOnly by nearly two times. Nevertheless, as node capability conforms to exponential distribution (like ***stl***), though GiantOnly only includes 26 (out of 200 nodes) giants in the DHT, the capacity ratio of GiantOnly to Grouping - $1/2$ on average - is far bigger than the ratio of the number of peers - $26/200$.

To contrast Grouping with Chord and GiantOnly in search efficiency, we assume the target file locates on each group member equally likely. It easily proves that the expected number of search hops in Chord and GiantOnly is $\log_2 N - 1$ and $\log_2 m - 1$, respectively. We simplify related settings in Grouping as follows:

1) each group is a k-regular graph ($k = 4$);
2) erasure coding is not implemented;
3) flooding search is performed for the target.

Based on the above settings, Grouping's unstructured search is simulated as below: 100 sample graphs are randomly generated for each group, and 100 random flooding is performed on each sample. Figure 7 plots the cases that the eight biggest groups are the destination groups, and for the sake of comparison, $\log_2 N - 1$ and $\log_2 m - 1$ are plotted, as well. It can be figured out that even in these large-size groups, the unstructured search only incurs slightly more hops, e.g., 2.77 in G_1, 0.99 in G_2, and much less in other groups. Henceforth, the overall search hops in Grouping is still less than that of Chord. Needless to say, this enhancement is closely related to our specific settings, and bandwidth might somehow suffer as a result of hop decrease. Nevertheless, taking into account that most groups are relatively small-size, the tradeoff between search latency and bandwidth consumption from a system perspective seems resolvable by delicately tuning relevant parameters on demand.

V. Conclusions and Future Work

In this paper, we employ the grouping strategy for the utilization of unstable nodes in high-churn DHT deployment. A homogeneous grouping scheme is proposed to enhance scalability. Not only are all nodes admitted to extend system storage capacity, but it also clusters homogeneous nodes together, deploys the inter- and intra-group connections distinctively, and tunes the number of groups, in an effort to ease search latency. How to maximize stability under this scheme is explored by formulating the problem Maximum Stability of Grouping. We prove it to be NP-hard, and elaborate on its infeasibility. Therefore, we propose an approximated grouping approach and reduce it to a feasible optimization problem. Simulation results reveal that Grouping derives a better stability-scalability tradeoff. Based on our measurement metrics, it is 4.1 times as stable as Chord with similar capacity and mild alleviation in search latency. Contrast to GiantOnly, it enjoys double capacity by slightly compromising stability and search efficiency.

Notice that the difference between N and $N(t)$ influences the tradeoff effect - thus, if $N(t)$ will greatly change (e.g., expansion) for a long period, the original grouping ought to change, as well. A natural solution is to re-estimate $N(t)$ and then re-group, which, however, involves considerable file movement. We propose a light-weight mechanism: each group G_k splits into two subgroups uniformly, i.e., both of their ***stl***s range from y_{k-1} to y_k. One of them is assigned a new ID and is treated as a new group. Such an adjustment leads the ***stl*** ranges to overlap each other, yet subgroups have performances similar to the original one. More work is needed to evaluate its performance.

Acknowledgments

We thank all reviewers for their helpful comments and Prof. Haiyan Cai for his insightful advices. This work is partly supported by China NSF grants (60673154, 60721002, 60825205), Jiangsu High-Tech Research project of China (BG2007039), China 973 project (2006CB303000), and NSF grants (CNS 0531410, CNS 0626240).

References

[1] "BitTorrent website," *http://www.bittorrent.com*.

[2] "Skype website," *http://www.skype.com*.

[3] "eDonkey website," *http://www.edonkey.com*.

[4] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for internet applications," *Proc. of SIGCOMM 2001*, vol. 31, no. 4, pp. 149–160, 2001.

[5] S. Saroiu, P. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems," *Proc. of MMCN*, 2002.

[6] P. Godfrey, S. Shenker, and I. Stoica, "Minimizing churn in distributed systems," *Proc. of SIGCOMM 2006*, pp. 147–158, 2006.

[7] 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," *Proc. of SOSP 2003*, pp. 314–329, 2003.

[8] C. Blake and R. Rodrigues, "High availability, scalable storage, dynamic peer networks: pick two," *Proc. of HotOS'03, Lihue (Kauai), Hawaii, USA*, 2003.

[9] 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," *Proc. of SIGCOMM 2005*, pp. 73–84, 2005.

[10] "Planetlab website," *http://www.planet-lab.org*.

[11] "KaZaA website," *www.kazaa.com/*.

[12] C. Wang and K. Harfoush, "On the stability-scalability tradeoff of DHT deployment," *INFOCOM 2007*, pp. 2207–2215, 2007.

[13] F. Wang, J. Liu, and Y. Xiong, "Stable peers: existence, importance, and application in Peer-to-Peer live video streaming," *INFOCOM 2008*, pp. 1364–1372, 2008.

[14] 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.

[15] P. Kirk, "RFC-Gnutella 0.6," *http://rfc-gnutella.sourceforge.net*.

[16] 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.

[17] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 2, pp. 219–232, 2004.

[18] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 2, pp. 24–36, 1997.

[19] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," *INFOCOM 2004*, vol. 1, 2004.

[20] J. Kangasharju, K. Ross, and D. Turner, "Optimizing file availability in peer-to-peer content distribution," *INFOCOM 2007*, pp. 1973–1981, 2007.

[21] G. Ausiello, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[22] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems," *INFOCOM 2004*, 2004.

[23] G. Pandurangan, P. Raghavan, and E. Upfal, "Building low-diameter P2P networks," *Proc. of FOCS 2001*, pp. 492–499, 2001.

[24] J. Little, "A simple proof of the queuing formula L= λ W," *Operations Research*, vol. 9, pp. 383–387, 1961.

Appendix

Algorithm 1: Get_τ_k_from_G_k

```
Data: $G_k = \{n_1, n_2, \ldots, n_{|G_k|}\}$
Result: $\tau_k$
begin
    Put $G_k$ into array $a[2|G_k|]$ in ascendant order;
    Suppose $a[0]$ is $n_t.joinTime$;
    $s \Leftarrow 0$;
    $pivot \Leftarrow 0$;
    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
                $s \Leftarrow s + (n_t.leaveTime - n_t.joinTime)$;
                $pivot \Leftarrow i + 1$;
                break;
            if $a[i]$ is a joining time then
                Push;
            else
                Pop;
    $\tau_k \Leftarrow s$;
end
```