Institute of Telematics

Technical Report

Optimal Node Splits in Hypercube-based Peer-to-Peer Data Stores

Dietrich Fahrenholtz

December 2006

Report No. **TR-2006-12-01**

Hamburg University of Technology
Schwarzenbergstrasse 95, 21073 Hamburg, Germany
phone: +49 40 42878-3531; Fax:+49 40 42878-2581
email: telematik@tuhh.de
web: http://www.ti5.tu-harburg.de

Optimal Node Splits in Hypercube-based P2P Data Stores

Dietrich Fahrenholtz[1], Volker Turau[1], and Andreas Wombacher[2]

[1] Hamburg University of Technology, Institute of Telematics, Germany
{fahrenholtz|turau}@tu-harburg.de
[2] Swiss Federal Institute of Technology, Dist. Information Sys. Lab., Switzerland
andreas.wombacher@epfl.ch

Abstract P2P data stores excel if availability of inserted data items must be guaranteed. Their inherent mechanisms to counter peer population dynamics make them suitable for a wide range of application domains. This technical report presents and analyzes the *split* maintenance operation of our P2P data store. The operation aims at reorganizing replication groups in case operation of them becomes sub-optimal. To this end, we present a formal cost model that peers use to compute optimal points when to run performance optimizing maintenance. Finally, we present experimental results that validate our cost model by simulating various network conditions.

1 Introduction

Peer-to-Peer network research over the last few years was mainly inspired by devising and proving novel Peer-to-Peer networks (P2PN) that exhibit good lookup performance [12], efficient routing tables [15], low maintenance overhead [16], or resilience against peer population dynamics [18] to quote a few research objectives. The traditional, research driving application domain of P2PN is file-sharing, while nowadays interest focuses more on other domains. Among these we find storage of reputation and trust information [17,5], support of digital libraries [19] or distributed variants of DNS [3], for example. These domains demand that data inserted be available. P2P data stores promise to satisfy this demand. They employ a distributed hash table (DHT) that maps a large key space onto a set of network nodes in a distributed and deterministic manner. Hash functions serve to generate search keys from data so that the latter can be identified uniquely and looked up efficiently (generally within $O(\log N)$ routing steps). However, performance of data access and data availability can be severely affected by peer population dynamics (a.k.a. *churn* or *fluctuation*). The *turnover rate* at which peers join or leave the P2P data store characterizes fluctuation. The latter can both threaten data availability and impede good performance. So every P2P data store has to take its own countermeasures against churn.

The paper is structured as follows: First, we introduce briefly our P2P data store and how peers organize to guarantee performance and cope with a growing peer population (sec's. 3 and 3.2). Second, we contribute a novel cost model used to derive optimal thresholds of number of peers in a group for a split that ensures the P2P data store keeps its performance (sec. 4). Then simulation experiments

show how to derive split thresholds from the model and confirm their optimality and adequacy for our P2P data store (sec. 5).

2 Related work

Every P2P network destined for real world deployment has to provide mechanisms that handle both concurrent and sustained peer joins and leaves while also keeping the network connected. There are relatively few papers addressing this issue in P2P data stores. Bamboo [18], for example, uses periodic maintenance to keep nodes' routing tables consistent thus ensuring good lookup performance in case of high peer population dynamics. The Kelips system [8] like Bamboo employs a periodic but, in contrast to the latter, epidemic-style communication to distribute membership information about peers. In these systems, a file-inserting peer must either periodically refresh its file to keep it from expiring (Kelips) or rely on a service outside the DHT (Bamboo). However, maintenance operations in both systems do not optimize system performance but rather ensure good operation. Our P2P data store has replication groups that facilitate maintenance and guarantee data availability by freeing data item inserting peers from the burden of refreshing their data, which is a substantial step towards P2P databases. Authors of [14] propose a P2P system that bears a strong resemblance to our P2P data store and thus also withstands high fluctuation. In their approach, maintenance operations called SPLIT and MERGE ensure routing tables do not become too large and data loss is avoided, respectively. However, they use simple formulas to compute split and merge thresholds, whereas we use an optimization approach to dynamically compute the best split thresholds. We think the solution presented in this paper naturally lends itself to extend their system.

3 Key ideas of our P2P data store

Due to space constraints we can only briefly present the key ideas of our P2P data store. More details can be found in [7]. First of all, a *P2P data store* is a subcategory of P2P networks. One of its distinctive features is that its members offer a data availability guarantee, i.e., data items once inserted will not be lost with high probability. Like in P2PN, Peers connect to a network infrastructure and collaborate to form an overlay network. Our P2P data store's overlay network resembles a hypercube topology that helps locate data items efficiently. They can be accessed and manipulated by basic DHT operations, namely *insert*, *update*, and *lookup*. Upon joining, a peer's local data store is empty. If it is not, the peer may not join our P2P data store. Note, as opposed to pure P2P networks, only P2P data stores guarantee successful discovery of previously inserted data items with configurable likelihood.

According to Leighton [13], a hypercube is a very powerful and well-understood network for parallel computation. This makes it ideal for P2P data stores and we decided to base ours on it as do authors of [14,18]. By definition, a hypercube of dimension d has $N = 2^d$ nodes, where each node is associated with an ID represented by a bit vector of size d. A node is connected with d other nodes via bidirectional links. In particular, there is a connection between a node

$u = u_1 \dots u_d$ and a node $v = v_1 \dots v_d$ if and only if their two IDs exactly differ in one bit, in which case we call the two nodes *acquainted*. Sending a message from node u to node v via such a link represents a *routing step*. However, this requires each peer to store a routing table that is made up of d references to remote nodes in different dimensions.

All peers act independently and autonomously of one another. So peers' natural behavior, i.e., joining and leaving the P2P network at one's own discretion, directly affects the availability of data items. In the worst case this would lead to data loss or, less worse, severe performance degradation because only few remaining peers would have to answer all requests. Our P2P data store employs the concept of *replication groups*, which are identical to hypercube nodes. Every group contains a number of peers that are *neighbors* of one another. They store and replicate data items of the node to which they belong, thus establishing availability for all that node's data items. Inter-group self-organization must take precautions against a shrinking peer population fluctuation. We analyzed this issue and proposed the coalesce maintenance operation that ensures data availability with configurable confidence in [4]. Peers can be in two distinct states: *active*, i.e., online and a member of the P2P data store, or *inactive* otherwise. Each node also has at least one *coordinator* that is responsible for bootstrapping joining peers, administering group data such as the group's neighbor list, and performing regular maintenance operations.

3.1 Data availability guarantee

The probability of data items being available and accessible is directly related to the number of active peers in each group. If there is at least one active peer per group, all data is available. On the other hand, if the last active peer of a group also leaves, all data items are lost. Peer populations are inherently volatile and thus a data availability strategy has to minimize the risk of complete data loss. If the probability of a peer being inactive is known and constant over a fixed period of time, one can easily solve the equation $\Pr[A \geq x] \geq a$ for the minimum number of active peers, x, while holding the confidence level, a, fixed. However, the probability of a peer being inactive is directly dependent on the peer population dynamics and thus can vary considerably.

3.2 Self-organization of the P2P data store

Every P2P network needs self-organization so that peers can issue and answer lookup, insert and update requests successfully despite fluctuation of the peer population. The primary goals of self-organization are

1. maintain data availability, P2P data store performance and overlay network cohesion and, at the same time,
2. keep maintenance bandwidth consumption as low as possible.

Notice however, the more bandwidth is spent on maintenance the better overlay network cohesion, but also the less bandwidth remains for data retrieval and manipulation operations. We devised two essential self-organization operations, i.e., *split* and *coalesce*. Both target the replication group level but have different

objectives. Technical details about how they operate can be found in [7]. The objective of the first maintenance operation, coalesce, is guaranteeing data availability, which means the number of peers of each group must not fall below a minimum threshold. If peers continuously leave the P2P data store and fusions never happen, this objective would be violated over the course of time and data loss would ensue. We analyzed this operation in [4] and showed how coordinators calculate the optimal point for a coalescence so that sibling groups do not loose data items with high probability. In contrast to a coalescence, a split operation is invoked to optimize the performance of the P2P data store. However, during a split data availability must not be threatened. A coordinator splits up its group into two sibling groups each getting one half of the peers. After that the parent group ceases to exist. Thus maintenance costs and request load on the former parent group are distributed among sibling groups. In the next sections we shall analyze this operation and give a cost model that is used by coordinators to compute the optimal point when to split up their groups.

4 Optimum for a split

Before we describe the process by which a coordinator arrives at a decision whether a split is optimal, we state prerequisites for our analysis. First of all, we require several conditions to hold throughout the lifetime of our P2P data store:

1. Coordinators involved in a split do not leave the network.
2. There is an upper bound on network packet transmission and processing delay.
3. Messages are transferred reliably.
4. Every peer can reach every other peer.

We further assume there are periods where the P2P data store grows so that eventually group splits become necessary. Over the course of time the following events can happen. They occur at their individual rate and are random by nature. The time between two events of the same type is called *interarrival time*. We define $E = \{join, leave, look, ins/upd, rt\_upd\downarrow, rt\_upd\uparrow, hb, idle\}$ as the set of all possible events.

Event	**Event rate**
a peer joins	λ_{join}
a peer leaves	λ_{leave}
a data item is looked up or insertions/updates	λ_{look}
a data item is inserted/updated	$\lambda_{\text{ins\_upd}}$
inbound and outbound routing table updates	$\lambda_{\text{rt\_upd}\downarrow}$
outbound routing table update	$\lambda_{\text{rt\_upd}\uparrow}$
heartbeats	λ_{hb}

Whenever a coordinator detects an event, it adapts the specific event rate whose unit of measurement is number of events per time unit. A group is idle if no event occurs. The first four event types directly depend on behavior of users operating peers and are thus inherently unpredictable and volatile, whereas the last three event types occur regularly. Each coordinator continuously monitors its efforts

to operate its group. Since peers may crash or leave the network without advice, coordinators employ a failure detector like the one proposed in [2]. It bases on peers sending heartbeat messages periodically to their coordinator. Two time outs are associated with each heartbeat. If the heartbeat times out for the first time, the peer is suspected to be inactive. If the second timeout occurs, it is considered inactive.

4.1 Operating groups and costs

Every group has to react to requests such as answering lookup requests, joining new peers, and so on. To this end, communication with other peers is a must. We call the number of bytes being transferred during communication the *costs* of an operation because peers need to "spend" bytes to receive and answer requests. The task of coordinators is to maintain their respective groups and refer to local data only when calculating group costs. Thus, extra communication is avoided. There are two types of costs incurred by group operations, i.e., operational costs and maintenance costs. The former costs are incurred continuously, whereas maintenance only incurs costs if either group operation is no longer optimal or data availability is threatened. Coordinators bear the main operational costs of a group, which are determined by joining or leaving a peer, updating peers' routing tables, and inserting or updating data items. The costs of an idle group are of course zero. Peers in a group bear the remaining operational costs, which are answering/forwarding lookup or insertion/update requests. Moreover, acknowledging the receipt of network packets also adds to operational costs. Furthermore, split and coalesce operations incur the second type of costs, i.e., maintenance costs.

Here we want to analyze the split maintenance operation invoked whenever it is better to distribute operational costs to two groups. First of all, we are giving cost functions for operational costs of coordinators. Operational costs separate into receiving and sending costs. By γ_{xxx} we denote constants that are fixed and do not change during lifetime of the P2P data store, whereas all other variables may change.

1. JOIN NEW PEER: $cost_{\text{join}}(C, R) := 7C + 7R + \gamma_{\text{pct}}$
   Explanation: When a new peer joins, it gets the list of all, C, active peers of the group it is about to join (first term) with the objective to become a neighbor of those peers. Each peer is uniquely identifiable by its IP address (4 bytes), port number (2 bytes), and meta data (1 byte) (e.g., peer is a coordinator or a backup coordinator, belongs to a particular bandwidth class, etc.). We call such a unique identification of a peer its *peer reference*. The second term is $R = \sum_{i=1}^{d} R_i$, where R_i is the number of references to peers in the acquainted node in dimension i in the routing table of the coordinator. Recall, the hypercube has d dimensions overall. Multiply this with the size of a peer reference to obtain the number of bytes necessary to transfer the complete routing table. To make a network message of this data ready for identification and transmission, additional γ_{pct} bytes of packet overhead are added. Finally, the coordinator increments C by one. Join costs belong to sending costs.

2. INSERT/UPDATE DATA ITEM: $cost_{\text{ins\_upd}}(C) := C(\gamma_{\text{data}} + \gamma_{\text{pct}})$
   Explanation: Peers of the penultimate routing step send data item insertion/update messages directly to coordinators of the destination node, which in turn spread these messages to every peer of their group so as to provide for data item replication. Insert/update costs belong to sending costs.
3. FAILURE DETECTION: $cost_{\text{fd}}() := \alpha + \gamma_{\text{pct}}$
   Explanation: Employing a failure detector has two advantages: 1) detection of crashed peers or network partitions is possible and 2) extra meta data such as individual peer statistics about forwarded but answered lookup messages are piggybacked on heartbeat messages. All peers send IAA ("I am alive?") heartbeat messages that consume bandwidth proportional to the current number of peers in the group. α gives the size of these meta data in bytes. Detecting peer failures belongs to receiving costs.
4. SEND NEIGHBOR LIST: $cost_{\text{rt\_upd}\uparrow}(C, d) := (d+1)(7C + \beta + \gamma_{\text{pct}})$
   Explanation: Every now and then a coordinator advises peers of its own group and peers of acquainted nodes of a new neighbor list because the composition of its group has changed due to new peers having joined or old ones having left. Out-dated routing table entries affect lookup performance and network cohesion in the long run. Outbound routing table updates correct this deterioration of routing tables. An update comprises of all C peers being active plus meta data about the group such as node ID, the number of data items stored in the group, and group statistics. The size of meta data is β bytes. Since d acquainted nodes plus the own group should get the new neighbor list, this makes a total of $d+1$ updates. Of course, sending neighbor lists belongs to sending costs.
5. RECEIVE AND FORWARD NEIGHBOR LIST: $cost_{\text{rt\_upd}\downarrow}(C, \overline{R}) := C(7\overline{R} + \beta + \gamma_{\text{pct}})$
   Explanation: Of course remote group coordinators also send routing table updates, but the number of peer references contained in each update usually varies. Receiving coordinators distribute routing table updates from acquainted nodes among all C peers of their group, so eventually every peer has an up-to-date routing table. To calculate forwarding costs, we take the average number of peer references, $\overline{R}$, of all d acquainted groups and add meta data and network packet overhead. In case an epidemic-style multicast [9] is employed, variable C can be replaced by some constant k denoting the initial number of peers to whom the routing table update is spread out. Of course, receiving neighbor lists belongs to receiving costs.

There are further operational costs such as the sending and receiving of acknowledgments of network packets that need to be taken into account. Notice, coordinators receive acknowledgments for packets they have sent and in turn acknowledge packets sent by other peers. We define the following cost function for acknowledgments

$$cost_{\text{ack}}(x) := \left\lceil \frac{x}{\gamma_{\text{pss}}} \right\rceil \cdot \gamma_{\text{pct}} \ ,$$

where γ_{pss} is the payload segment size in bytes which defines the maximum number of application bytes to fit in one network packet.

In section 4 we defined events that may happen in a group. Since we have recurring events and two consecutive event rates can deviate considerably from

each other, the decision to split up a group should be better based on several recorded past event rates to smooth volatility. To this end, we let coordinators calculate an *n-moving average*

$$\overline{\lambda_{\text{join}}} = \sum_{j=i}^{i+n-1} \frac{\lambda_{\text{join}}^{j}}{n} \quad n, i > 0 \tag{1}$$

where $\{\lambda_{\text{join}}^{1}, \ldots, \lambda_{\text{join}}^{n}\}$ is a sequence of n consecutive join event rates. When the $n+1^{\text{st}}$ event rate is added to the sequence, pointer i shifts one position forward. Calculation of moving averages for other event types is done analogously. A moving average is very popular in finance and serves as a trend indicator. An n-moving average has the advantage that consequences of highly volatile samples on the moving average are mitigated if n is chosen large. Conversely, if a small value for n is used, the reaction to movements is much more immediate.

When multiplying individual costs with average event rates, respectively, we obtain net costs coordinators have to spend. Cost figures are updated whenever a new peer joins a group. We give sending and receiving operational costs first.

$$\begin{aligned} cost'_{\text{opr}\uparrow}(C, R, \overline{R}, d) &:= \overline{\lambda_{\text{join}}} \cdot cost_{\text{join}}(C, R) \; + \overline{\lambda_{\text{ins\_upd}}} \cdot cost_{\text{ins\_upd}}(C) \; + \\ &\quad \overline{\lambda_{\text{rt\_upd}\uparrow}} \cdot cost_{\text{rt\_upd}\uparrow}(C, d) \\ cost'_{\text{opr}\downarrow}(C, R, \overline{R}, d) &:= \overline{\lambda_{\text{hb}}} \cdot cost_{\text{fd}}() \; + \overline{\lambda_{\text{rt\_upd}\downarrow}} \cdot cost_{\text{rt\_upd}\downarrow}(C, \overline{R}) \end{aligned} \tag{2}$$

Finally, we add acknowledgment costs to obtain total costs.

$$\begin{aligned} cost_{\text{opr}\uparrow}(C, R, \overline{R}, d) &:= cost'_{\text{opr}\uparrow}(C, R, \overline{R}, d) + cost_{\text{ack}}(cost'_{\text{opr}\downarrow}(C, R, \overline{R}, d)) \\ cost_{\text{opr}\downarrow}(C, R, \overline{R}, d) &:= cost'_{\text{opr}\downarrow}(C, R, \overline{R}, d) + cost_{\text{ack}}(cost'_{\text{opr}\uparrow}(C, R, \overline{R}, d)) \end{aligned} \tag{3}$$

Peers that are not coordinators have operational costs, too. Their costs should influence the point in time when to split up the group they belong to. Whenever peers answer lookup requests that pertain to data items they store, their individual operational costs increase. On the other hand, they forward requests they cannot answer. We do not want request forwarding to contribute to peers' operational costs because it does not concern data items stored in forwarding peers. Every split of a group extends the routing paths of those requests that target this group by one thus increasing request latencies. We think if peers in a group have to answer many lookup requests on average, splitting up this group should be postponed until request rate subsides. A penalty function, $\mathcal{P}(\overline{\lambda_{\text{look}}})$, that grows with the number of answered lookup requests meets this requirement. We have

$$\mathcal{P}(\overline{\lambda_{\text{look}}}) := \zeta \cdot \overline{\lambda_{\text{look}}} \cdot (\gamma_{\text{data}} + \gamma_{\text{pct}}) \; . \tag{4}$$

The sum $\gamma_{\text{data}} + \gamma_{\text{pct}}$ denotes the number of bytes of an answer. $\overline{\lambda_{\text{look}}}$ is the current average lookup rate of all C peers in a group and $\zeta \geq 1$ is a scale factor fixed at the start of the P2P data store. In fact, the penalty function shifts the point when to split so that additional peers can join the group thus eventually diminishing the load of answering lookup requests.

4.2 Group maintenance and costs

In this section we describe how the split maintenance operation works. We give details of the individual parts of the operation and the costs incurred. Coordi-

nators are the sole entities to decide whether their groups are to be split up or not.

1. SEND NEW BOUNDS AND NEIGHBOR LISTS: $cost_{\text{nb\_nl}}(C) := \frac{C}{2}\left(\frac{7C}{2} + \delta + \gamma_{\text{pct}}\right)$
   Explanation: A split divides an existing group into two new sibling groups having $\lceil \frac{C}{2} \rceil$ and $\lfloor \frac{C}{2} \rfloor$ number of peers, respectively. Peers of both groups must know about their new neighbors, coordinator, and backup coordinators. This information amounts to $\frac{7C}{2}$ bytes per group. Later on, routing table updates inform about which peers are in the other sibling group. During split up, peers also get new bounds that shift their responsibility for data items from $[a..b]$ to $\left[a..\lfloor \frac{a+b}{2} \rfloor\right]$ for the left and $\left[\lceil \frac{a+b}{2} \rceil..b\right]$ for the right sibling group, where a and b are the lower and upper bound of the search key range of the old group. Sending bounds costs δ number of bytes. These costs arise twice (once for each group). At the end of a split, peers increment their dimension counter by one and drop data items they are no longer responsible for.
2. DETERMINE NEW COORDINATORS: $cost_{\text{nc}}() := \gamma_{\text{stats}} + \gamma_{\text{pct}}$
   Explanation: Every coordinator and its backup coordinators keep their roles when changing from an old to a new group, thus there are no costs incurred by finding new ones. In the previous step, the old coordinator has also determined the coordinator and at least one backup coordinator of the other sibling group and has sent this information to all peers in this group. The task of new backup coordinators is to advise one another and the new coordinator of their active presence immediately. Once the new coordinator knows that at least one backup coordinator is active, it declares the new group operational and advises the old coordinator accordingly, which in turn answers with group statistics comprising of γ_{stats} bytes. If backups do not get any response from the new coordinator, the backup with the highest IP address that is still active becomes new coordinator and checks that at least one other backup is active. We consider the probability of a new coordinator and all backups being inactive simultaneously to be negligible so these costs are incurred only once.

After a split, a routing table update advises acquainted nodes of the two new groups. Thus, total split costs sum to

$$\begin{aligned} cost_{\text{split}}(C) := 2 \cdot cost_{\text{nb\_nl}}(C) + cost_{\text{nc}}() + cost_{\text{rt\_upd}\uparrow}(C, d-1) + \\ cost_{\text{ack}}(2 \cdot cost_{\text{nb\_nl}}(C) + cost_{\text{nc}}() + cost_{\text{rt\_upd}\uparrow}(C, d-1)) \ . \end{aligned} \tag{5}$$

Notice the split maintenance operation makes sure the hypercube remains balanced, i.e., there is no more than one dimension difference between acquainted nodes. If joining a new peer were to lead to a split that would imbalance the hypercube, the peer is turned down and has to try a different group for re-joining. Thus the P2P data store's routing structure remains balanced.

4.3 Split model

We are now ready to formulate a model to be solved by Nonlinear Programming, an Operations Research tool [11], that helps coordinators decide when a split is

optimal. To this end we, first, introduce parameters of the model. We let

$$\frac{cost_{\mathrm{opr}\uparrow}(C,R,\overline{R},d)+cost_{\mathrm{opr}\downarrow}(C,R,\overline{R},d)-\mathcal{P}(\overline{\lambda_{\mathrm{look}}})}{cost_{\mathrm{split}}(C)+cost_{\mathrm{opr}\uparrow}(C',R',\overline{R'},d+1)+cost_{\mathrm{opr}\downarrow}(C',R',\overline{R'},d+1)} \tag{6}$$

the objective function to be maximized subject to constraints

$$cost_{\mathrm{opr}\uparrow}(\cdots) < B_{\mathrm{C}\uparrow} \ , \tag{7}$$
$$cost_{\mathrm{opr}\downarrow}(\cdots) < B_{\mathrm{C}\downarrow} \ , \tag{8}$$
$$\lfloor C/2 \rfloor > C_{\min} \ , \tag{9}$$
$$\lambda_{\mathrm{join}} - \lambda_{\mathrm{leave}} > 0 \ . \tag{10}$$

Here $C_{\min}$ is a lower bound of peers a group resulting from a split must contain to guarantee data availability also taking into account the *group size change rate* defined by constraint (10) might become negative after the split. Values of $C_{\min}$ are unique for each group and coordinators, with a method presented in [4], reevaluate their prediction of $C_{\min}$ whenever an event in their groups occurs. The denominator of eq. (6) estimates costs of one of the sibling groups resulting from a split plus the split costs. However, those costs need adapted event rates. Specifically, join rate, heartbeat rate, and insert/update rate halve on average. This is because events occur uniformly at random over the complete search interval and each new group assumes responsibility for half of the parent's group search interval. However, the number of future routing table updates in both directions will increase by one per update period because such updates must also reach the other sibling group. New sibling groups contain $C' = \frac{C}{2}$ peers and have $R' = R + \frac{C}{2}$ peer references overall in their routing table leading to $\overline{R'} = \frac{d\overline{R}+C/2}{d+1}$ peer references on average.

Peers can have an asymmetric network connection (e.g., ADSL) and we take this into account by introducing $B_{\mathrm{C}\uparrow}$, which is the available sending bandwidth of a coordinator, and $B_{\mathrm{C}\downarrow}$, which denotes its available receiving bandwidth. Altogether, if the ratio in (6) is maximal and all constraints hold, then this defines the best point when to split a group. Constraints (7) and (8) have special importance. If one or both of them become false and constraint (10) is true but constraint (9) is not, a coordinator must hand over the coordinator role to a different peer in its group that is more capable in terms of bandwidth.

Notice that variables C, R, and all event rates are random and thus there is no single optimal solution that can be analytically derived. Thus this model needs to be validated with simulations.

Remarks to the split model. The approach shown above actually belongs to the family of *fractional programming* models. The latter are used, for example, when one wants to find the maximum productivity, i.e., the ratio of output to man-hours expended, or maximum rate of return, i.e, profit to capital expended, or maximize the return vs. risk ratio, i.e., expected value to standard deviation of some measure of performance for an investment portfolio (cf. [11]). When we set out to formulate cost model, our first approach was a non-linear quadratic

programming model. Our objective function to be maximized was

$$\begin{aligned} &cost_{\mathrm{opr}\uparrow}(C, R, \overline{R}, d) + cost_{\mathrm{opr}\downarrow}(C, R, \overline{R}, d) - \mathcal{P}(\overline{\lambda_{\mathrm{look}}}) < \\ &2(cost_{\mathrm{opr}\uparrow}(C', R', \overline{R'}, d+1) + cost_{\mathrm{opr}\downarrow}(C', R', \overline{R'}, d+1)) + cost_{\mathrm{split}}(C) \ , \end{aligned} \tag{11}$$

while retaining the constraints mentioned above. We evaluated this model by testing it with various values for C, R and d in Maple [10]. However, validation of the model did not work, because either operational costs of parent groups always exceeded the left side of equation (11), or the model gives reasonable values for only a limited range of dimensions, d. We finally formulated our cost model in terms of a fractional programming problem and again evaluated it in Maple. It turned out that the ratio of operational costs minus the penalty to operational costs of a sibling group plus split costs is maximizable and gives valid and reasonable split thresholds across a wide range of values for C, R, and d as we will show in the next section.

5 Experiments for split threshold determination

In this section, we briefly describe our simulator and the experiments that show how the optimization model behaves and that the computed split thresholds are practical in various scenarios. One simulation variable such as the number of groups, number of data items per group, turnover rate, etc. is controllable in each experiment while the others remain fixed. So the dependency of this parameter on split thresholds can be checked. Finally, we want to quantify parameter ζ such that the penalty function influences the computation of split thresholds adequately when varying lookup rates.

5.1 Features of our simulator

We developed an activity-based simulator written in Java that allows concurrent, non-deterministic peer interactions. We found this approach more suitable than a pure discrete event simulation, a view that is also supported by authors of [1]. Every peer is associated with a thread and interacts with others through objects providing a message channel abstraction. There are configuration directives such as number of coordinators per group, maximum group size, etc. to set up our P2P data store. Network parameters such as packet failure percentages and latencies can also be adjusted. Moreover, the type, the frequency in percent, and the total number of data store layer operations that are to occur in a simulation phase can be set. Finally, the mean interarrival time between two consecutive data store layer operations and the configuration of how statistics should be gathered can be tuned.

Simulation setup. Essentially, in each experiment the peer population grows at a specific positive group size change rate. Peers join hypercube nodes uniformly at random. Once a new peer has joined, the joining coordinator employs a search procedure to find the current split threshold that maximizes the objective function. If the number of peers in its group reaches this optimum and

all other constraints hold, then a split is performed. We record and aggregate all split thresholds of groups in the same dimension. Moreover, all simulations are separated into three phases. First of all, a minimum number of peers joins the initially empty P2P data store. This leads to a number of group splits until the hypercube contains a target number of nodes. Second, a phase in which a phase-specific number of distinct data items are spread uniformly about existing groups follows. This phase also stabilizes the P2P data store, i.e., the average turnover rate, reaches zero. Finally, the third phase grows the overall peer population at a phase-specific positive group size change rate. Fluctuation affects all groups uniformly on average. However, individual group turnover rates can differ from one another. Furthermore, all activity starts are Poisson distributed and the time unit is 10 seconds for all experiments. So we make sure coordinators can react in a timely manner on peer population changes. One part of following parameter values comes from the application domain *reputation management* [5], while the other parameter values stem from Internet traffic measurements. We set the size of a data item, $\gamma_{\text{data}} = 200$ bytes, the packet overhead, $\gamma_{\text{pct}} = 48$ bytes, additional meta-data when advising acquainted nodes, $\beta = 150$ bytes, and $\gamma_{\text{pss}} = 1450$ bytes. Constants needed for calculating split costs are: $\delta = 40$ bytes and $\gamma_{\text{stats}} = 660$ bytes. The latter constant sums sizes of n event rate samples plus lookup and group meta data. We record $n = 10$ consecutive samples for each event type making event histories 100 seconds long. Values for $C_{\min}$ are predicted during runtime. Every peer is required to send a heartbeat to its coordinator every 3 time units.

5.2 Simulation results

For the first string of experiments, we wanted to study the behavior of our model, i.e., the change to split thresholds when varying the number of groups. We set the number of data items per group to 0, $\lambda_{\text{look}} = 0$ and the average overall turnover rate is set to 10 peers per time unit. Figure 1 shows error bars that indicate minimum, average, and maximum split thresholds for P2P data store dimensions from 0 (1 group) to 8 (256 groups) using the lower x-axis. Two things are striking: First, an increase in the number of dimensions leads to an increase in deviations of split thresholds. This is because the number of peers in each group is binomially distributed and the variance of this distribution increases linearly with the total number of peers in the P2P data store. Second, average split thresholds decrease toward dimension 3 but increase again afterwards. This is because the turnover rate halves from dimension x to dimension $x + 1$ and is highest in dimension 0. Only after the first time unit has passed, evaluation of the objective function is possible. If a group experiences a high turnover rate, the number of freshly joined peers can easily exceed the optimum split threshold during the very first time unit. This is the case for splits in the first two dimensions which occurred after the first time unit has passed. In higher dimensions, the increase in average split thresholds is because operational costs of groups in dimensions > 3 initially grow faster than the sum of operational costs of a new sibling group plus split costs. Finally, the second curve of figure 1 shows safety margins using error bars and the upper x-axis. These margins capture the difference between split thresholds (first curve) and $C_{\min}$ and grow

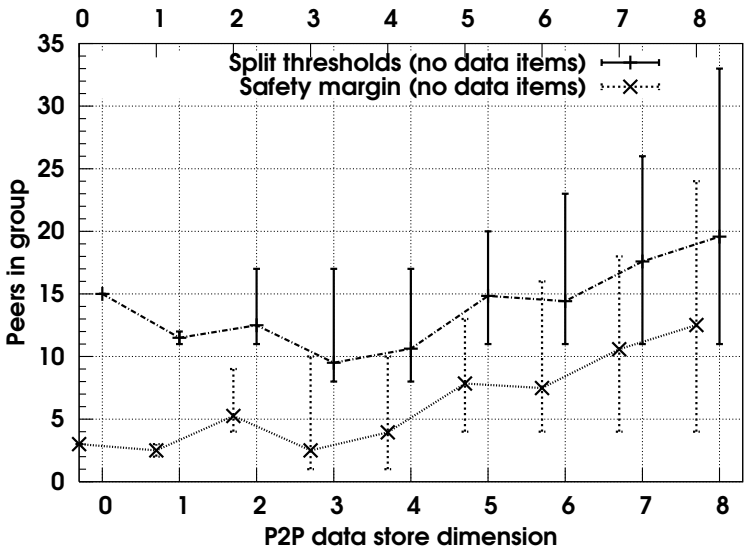

Figure 1. Simulation results varying number of groups

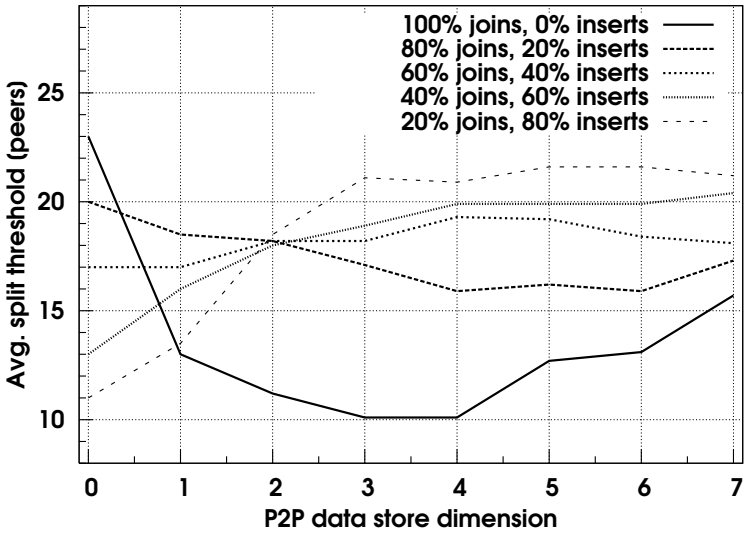

Figure 2. Simulation results varying the joins to inserts ratio

with the size of the P2P data store. If a safety margin is small and the group size change rate turns negative after a split, for example, then sibling groups have to coalesce all the earlier the smaller the margin. In effect, this means if the size of our Peer-to-Peer data store is rather small (below 4 dimensions), it is most vulnerable to fluctuation.

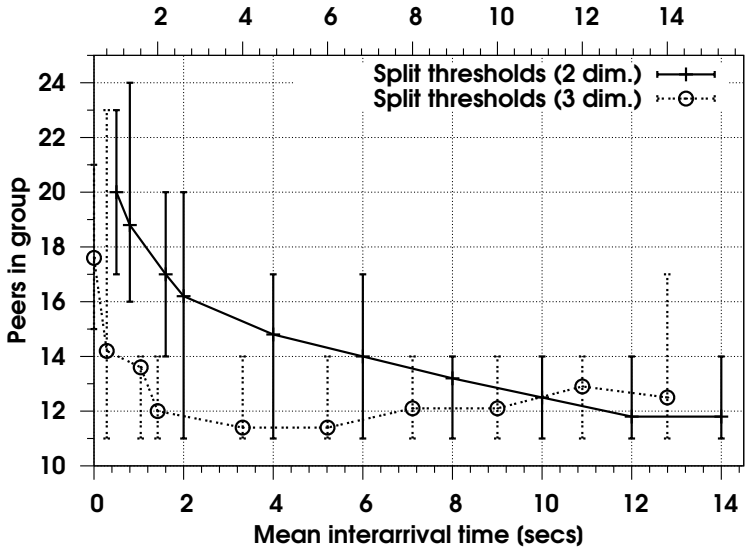

Figure 3. Simulation results varying interarrival time of turnover events

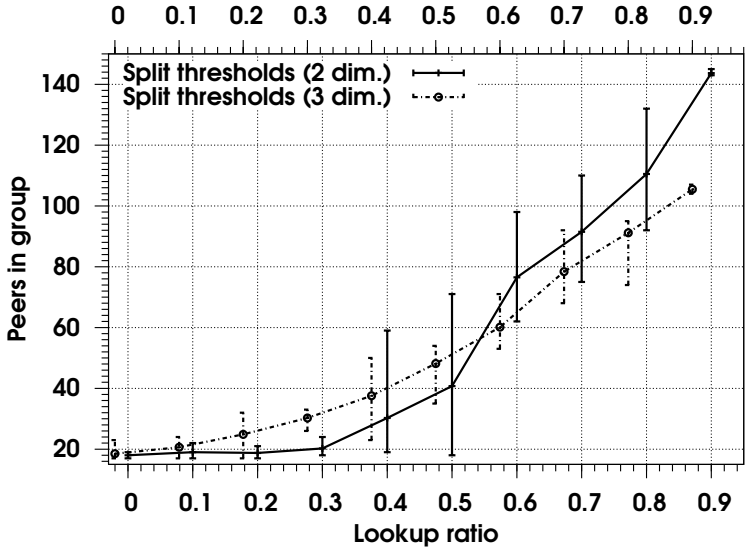

Figure 4. Simulation results varying lookup/turnover events ratio

Next we investigated how a varying ratio of data item insertions to peer joins affects average split thresholds, while keeping average event rate fixed. Initially, our P2P data store is empty. Only after the first peer has joined successfully, data item insertions occur. Insertions and joins occur randomly with ratios 100:0, 80:20, 60:40, 40:60, and 20:80, where the left-hand side of the ratio denotes the join percentage and the right-hand side denotes the insert percentage. Both are

fixed at the beginning of each experiment. Note, a ratio of 0:100 would not make sense because coordinators perform splits only if new peers join their group. We simulate P2P data store growth up to 7 dimensions. All other parameters remain as mentioned before, except for the turnover rate, which was set to 2, i.e., two peers join every second on average. Figure 2 shows average split thresholds, which are connected by lines so as to make clear which thresholds belong to which ratio. If no data items insertions occur, we observe the same phenomenon as for the first series of experiments: a high split threshold when the first group splits and a quick decrease of thresholds up to dimensions 3 and 4. The reason is the same as for the previous experiments. Overall one can observe that curves change from rather convex to rather concave if the insert percentage increases and the join percentage decreases. Recall that the term $\overline{\lambda_{\text{ins\_upd}}} \cdot cost_{\text{ins\_upd}}(C)$ in equation 2 denotes the average current costs of a coordinator to forward data item insert/update operations to its peers and $\overline{\lambda_{\text{join}}} \cdot cost_{\text{join}}(C, R)$ in the same equation denotes the average current costs to join a new peer. Also note in lower dimensions average insert/update costs are higher than average costs to join a new peer. So the more insert/update operations a coordinator must forward the more work on top of joining new peers it has to do, which adds to current operational costs. Since all event rates per group halve every additional hypercube dimension, this effect is especially noticeable when the hypercube contains only few dimensions, say, up to 3. Figure 2 hints that individual split thresholds for a join/insert ratio converge in the limit $d \to \infty$, since both join and insert percentage become zero.

The effects of varying mean interarrival times of turnover events on split thresholds are shown in figure 3. For these experiments, we let our P2P data store grow to exactly 4 groups and subsequently inserted 1000 data items uniformly at random. Then our P2P data store grows to up to 5 dimensions and while growing we record split thresholds from groups in a 2-dimensional (lower x-axis) and 3-dimensional (upper x-axis) hypercube. Error bars again express the minimum, average, and maximum split thresholds of groups in the same dimension. We see a decrease of average split thresholds down to lower bounds in the 2-dimensional and 3-dimensional hypercubes, respectively. This was expected and happens because decreasing the turnover rate also decreases average operational costs. Since the turnover rate in the 3-dimensional hypercube is lower than the rate in the 2-dimensional one, split thresholds of the former must also be lower. The decrease in the 3-dimensional hypercube, however, reaches its lower bound earlier than in the 2-dimensional hypercube due to the turnover rate being half of the rate in the 2-dimensional hypercube. When the mean interarrival time is 14 secs., we can see an outlier for the maximum split threshold in the 3-dimensional hypercube. It was produced by a group whose coordinator had too little bandwidth to do the split. In the meantime when a new more capable coordinator had to be determined and established, new peers joined the group so that split threshold became higher than the others. Also notice, if the mean interarrival time of events in a group is shorter than 200 ms, no split will be performed. This is because constraint (9) never holds.

In the last string of experiments, we varied the ratio of lookups to turnover events from 0 to 0.9 to find a good value for variable ζ used in the penalty function. A good value is found if the average split threshold curves are not too steep

and not too flat over a wide range of P2P data store sizes. All other parameters and the meaning of error bars remained as before. The average interarrival event time was set to 0.5 time units. Otherwise, the P2P data store was prepared as for the previous experiments. Figure 4 depicts the increase of split thresholds for a 2- (lower x-axis) and 3-dimensional (upper x-axis) P2P data store. It clearly shows the greater the ratio lookup events to turnover events the more peers a group has to accumulate to be allowed to split. We experimented with various values for ζ and found out that $\zeta = 9$ leads to best results for a wide range of dimensions.

6 Conclusion

In this paper we have introduced and analyzed the maintenance operation 'split' whose task is to optimize the performance of groups in our P2P data store. We developed a Nonlinear Programming cost model and validated it by simulations. The simulation results show that sensible optimal split thresholds can be computed. Moreover, split thresholds computed in this way much better take different lookup rates, number of data items stored in peers, etc. into account as would split thresholds that simply depend on the logarithm of the number of peers in the P2P data store "eQuus" [14]. Also important, there is always enough room between split thresholds and minimum numbers of peers necessary for a fusion, which means the coordinator of a new sibling groups would not been forced to initiate a coalesce operation directly if after the split peers suddenly start to leave its group or crash. Thanks to the penalty function, coordinators are able defer splits if their groups are the destination for many requests. However notice, maintenance operations ought to be invoked infrequently because of their bandwidth consumption.

The evolution of the peer population in file sharing P2PN follows a circadian pattern [20]. That means the frequency at which splits need to be performed will be low if the P2P data store is so large and has so many groups as to take the peer population dynamics over a day on average.

References

1. R. von Behren, J. Condit, and E. Brewer. Why Events are a Bad Idea (for High-concurrency Servers). In *Proc. of the 9th Workshop on Hot Topics in Operating Systems*, 2003.
2. M. Bertier, O. Marin, and P. Sens. Implementation and Performance Evaluation of an Adaptable Failure Detector In *Proc. of the International Conference on Dependable Systems and Networks*, Bethesda, MA, USA, 2002.
3. R. Cox, A. Muthitacharoen, and R. T. Morris. Serving DNS Using a Peer-to-Peer Lookup Service. In *Proc. of the 1st Int'l Workshop on Peer-to-Peer Systems*, 2002.
4. D. Fahrenholtz and V. Turau Improving Churn Resistance of P2P Data Stores Based on the Hypercube. In *Proc. of the 5th Int'l Symposium on Parallel and Distributed Computing*, 2006.
5. D. Fahrenholtz and W. Lamersdorf. Transactional Security for a Distributed Reputation Management System. In *Proc. of the 3rd Int'l Conference on Electronic Commerce and Web Technologies*, 2002.

6. D. Fahrenholtz, V. Turau, and A. Wombacher. Optimal Node Splits in Hypercube-based Peer-to-Peer Data Stores. Technical Report TR-2006-12-01, Hamburg University of Technology, 2006.
7. D. Fahrenholtz and V. Turau. A Tree-based DHT Approach to Scalable Weakly Consistent Data Management. In *Proc. of the 1st Int'l Workshop on P2P Data Management, Security and Trust*, 2004.
8. I. Gupta, K. Birman, P. Linga, et al.. Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead. In *Proc. of the 2nd Int'l Workshop on Peer-to-Peer Systems*, 2003.
9. J. Hatje. *Datenreplikation mit probablistischen, epidemischen Verteilungsalgorithmen*. Diploma Thesis, Institute of Telematics, Hamburg University of Technology, 2005.
10. A. Heck. *Introduction to Maple*. 3rd. edition, Springer Verlag New York, 2003.
11. F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research*. 5th ed., McGraw-Hill Inc., 1990
12. F. Kaashoek and D. Karger. Koorde: A simple Degree-optimal Hash Table. In *Proc. of the 2nd Int'l Workshop on Peer-to-Peer Systems*, 2003.
13. F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, San Mateo, USA, 1992.
14. Th. Locher, S. Schmid, and R. Wattenhofer. eQuus: A Provably Robust and Locality-Aware Peer-to-Peer System. In *Proc. of the 6th IEEE International Conference on Peer-to-Peer Computing*, Cambridge, UK, 2006.
15. D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *Proc. of the 21st ACM Symposium on Principles of Distributed Computing*, 2002.
16. G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small World. In *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, USA, 2003.
17. S. Marti and H. Garcia-Molina. Taxonomy of Trust: Categorizing P2P Reputation Systems. Computer Networks 50(4): 472–484, 2006.
18. S. Rhea, D. Geels, et al.. Handling Churn in a DHT. In *Proc. of the USENIX Annual Technical Conference*, Boston, MA, USA, 2004.
19. T. Risse and P. Knesevic. A Self-organizing Data Store for Large-scale Distributed Infrastructures. In *Int'l Workshop on Self-Managing Database Systems*, 2005.
20. D. Stutzbach and R. Rejaie. Towards a Better Understanding of Churn in Peer-to-Peer Networks. Tech. Rep. CIS-TR-04-06, University of Oregon, 2004.