On Optimal Replication Group Splits in P2P Data Stores Based on the Hypercube

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 paper 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 [7], low maintenance overhead [9], or resilience against peer population dynamics [11] to quote a few research objectives. Today the application domains that drive research focus on storage of reputation and trust information [10] or distributed variants of DNS [1], 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. However, performance of data access and data availability can be severely affected by peer population dynamics (aka. *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 (section 2 and 2.1). 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 (section 4). Then simulation experiments show how to derive split thresholds from the model and confirm their optimality and adequacy for our P2P data store (section 5).

2 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 [4]. Peers connect to a network infrastructure and collaborate to form an overlay network with hypercube topology that helps locate data items efficiently. Data items can be accessed and manipulated by basic DHT operations, namely *insert*, *update*, and *lookup*. Upon joining, a peer's local data store is empty. Note, as opposed to pure P2P networks, only P2P data stores guarantee successful discovery of previously inserted data items with configurable likelihood.

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 two nodes if their bit vectors exactly differ in one bit in which case we call the two nodes *acquainted*. Each peer stores 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 data store 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 guaranteeing data availability for these data items. Inter-group self-organization must take precautions against a shrinking peer population. We analyzed this issue and proposed the coalesce maintenance operation that ensures data availability with configurable confidence in [2]. 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.

2.1 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. However notice, the more bandwidth is spent on maintenance the better overlay network cohesion, but also the less bandwidth remains for data manipulation and retrieval 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 [4]. The objective of the coalesce operation is to preserve data availability which is equivalent to the number of peers of each group must not fall below a minimum threshold. If peers continuously leave the P2P data store and fusions of nodes never happen, this objective would be violated over the course of time and data loss would ensue. We analyzed this operation in [2] and showed how coordinators calculate the optimal point for coalescence so that sibling groups do not loose data items with high probability. In contrast to a coalescence, a split is invoked to optimize performance while still guaranteeing data availability. To this end, a coordinator splits up its group into two sibling groups each containing half of the peers and retires the parent group. Thus maintenance costs and request load are distributed. 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.

3 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 [11], 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 [5] 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. Instead, in our P2P data store, replication groups maintain and guarantee data availability and thus free inserting peers from the burden of refreshing their data, which is a substantial step towards P2P databases. Authors of [8] 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 do not use an optimization approach as we do to compute best split thresholds. We think the solution presented in this paper naturally lends itself to also optimize their system.

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 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. Moreover, events such as a peer joins or leaves occur at their individual rate (λ_{join}, λ_{leave}) and are random by nature. The time between two events of the same type is called *interarrival time* being measured by coordinators. A group is idle if no event occurs. Events depending on user behavior such as insertions, updates, and lookup operations are unpredictable, whereas other events such as routing table updates occur regularly as do heartbeats, which are sent from peers to their coordinator to indicate their liveliness.

Every group has to react to requests such as answering lookup requests, joining new peers, etc. To this end, communication with other peers is indispensable. 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 any coordinator is to maintain its group and access local data only when calculating group costs. Thus, extra communication is avoided. In this paper we analyze the split maintenance operation invoked whenever it is better to distribute operational costs to two groups.

4.1 Operating groups and costs

Next, we give cost functions for operational costs, which are separated into receiving and sending costs. For a detailed explanation concerning the individual costs functions see [3]. γ_{pct} denotes the constant overhead to packetize a request and γ_{data} is the constant size of a data item. α and β give the size of additional meta data. Variable C is the current number of active peers in a group, R is the sum of all peer references currently stored in the routing table of that group's coordinator, and d is the current dimension of the group. Moreover, 7 bytes are necessary to uniquely identify a peer.

1. Join new peer: $cost_{\text{join}}(C, R) := 7C + 7R + \gamma_{\text{pct}}$
2. Insert/update data item: $cost_{\text{ins\_upd}}(C) := C(\gamma_{\text{data}} + \gamma_{\text{pct}})$
3. Failure detection: $cost_{\text{fd}}() := \alpha + \gamma_{\text{pct}}$
4. Send neighbor list: $cost_{\text{rt\_upd}\uparrow}(C, d) := (d + 1)(7C + \beta + \gamma_{\text{pct}})$
5. Receive and forward neighbor list: $cost_{\text{rt\_upd}\downarrow}(C, \overline{R}) := C(7\overline{R} + \beta + \gamma_{\text{pct}})$

When multiplying individual costs with average event rates (denoted by $\overline{\lambda_{\text{xxx}}}$), 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.

$$\begin{aligned} cost'_{\text{opr}\uparrow}(C, R, \overline{R}, d) &:= \overline{\lambda_{\text{join}}} \cdot cost_{\text{join}}(C, R) \ + \overline{\lambda_{\text{ins\_upd}}} \cdot cost_{\text{ins\_upd}}(C) \ + \\ &\qquad \overline{\lambda_{\text{rt\_upd}\uparrow}} \cdot cost_{\text{rt\_upd}\uparrow}(C, d) \\ cost'_{\text{opr}\downarrow}(C, R, \overline{R}, d) &:= \overline{\lambda_{\text{hb}}} \cdot cost_{\text{fd}}() \ + \overline{\lambda_{\text{rt\_upd}\downarrow}} \cdot cost_{\text{rt\_upd}\downarrow}(C, \overline{R}) \end{aligned} \tag{1}$$

Finally, we add costs for acknowledging requests to obtain total costs.

$$\begin{aligned} cost_{\text{opr}\uparrow}(C, R, \overline{R}, d) &:= cost'_{\text{opr}\uparrow}(C, R, \overline{R}, d) + cost_{\text{ack}}(cost'_{\text{opr}\downarrow}(C, R, \overline{R}, d)) \\ cost_{\text{opr}\downarrow}(C, R, \overline{R}, d) &:= cost'_{\text{opr}\downarrow}(C, R, \overline{R}, d) + cost_{\text{ack}}(cost'_{\text{opr}\uparrow}(C, R, \overline{R}, d)) \end{aligned} \tag{2}$$

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 routing paths of a number of lookup requests by one thus causing higher latencies for these requests. We think if peers in a group have to answer many lookup requests on average, splitting up this group should be postponed until lookup request rate decreases. A penalty function, $\mathcal{P}(\overline{\lambda_{\text{look}}}) := \zeta \cdot \overline{\lambda_{\text{look}}} \cdot (\gamma_{\text{data}} + \gamma_{\text{pct}})$ that grows with the number of answered lookup requests meets this requirement. 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.

4.2 Group maintenance and costs

Of course, group maintenance also incurs costs and we give split costs here. For further details, see [3]. In what follows, δ and γ_{stats} denote the number of bytes for sending bounds and group statistics, respectively.
1. Send new bounds and neighbor lists: $cost_{\text{nb\_nl}}(C) := \frac{C}{2}\left(7\frac{C}{2} + \delta + \gamma_{\text{pct}}\right)$
2. Determine new coordinators: $cost_{\text{nc}}() := \gamma_{\text{stats}} + \gamma_{\text{pct}}$
After a split, a routing table update advises acquainted nodes of the two new sibling groups. Thus, total split costs sum to

$$\begin{aligned} cost_{\text{split}}(C) := {} & 2 \cdot cost_{\text{nb\_nl}}(C) + cost_{\text{nc}}() + cost_{\text{rt\_upd}\uparrow}(C, d-1) + \\ & cost_{\text{ack}}(2 \cdot cost_{\text{nb\_nl}}(C) + cost_{\text{nc}}() + cost_{\text{rt\_upd}\uparrow}(C, d-1)) \ . \end{aligned} \tag{3}$$

Notice, the split maintenance operation makes sure the hypercube remains balanced, i.e., there is no more than one dimension difference between acquainted nodes.

4.3 Split model

We are now ready to formulate a model to be solved by Nonlinear Programming, an Operations Research tool [6], that helps coordinators decide when a split is optimal. To this end we, first, introduce parameters of the model. We let

$$\frac{cost_{\text{opr}\uparrow}(C, R, \overline{R}, d) + cost_{\text{opr}\downarrow}(C, R, \overline{R}, d) - \mathcal{P}(\overline{\lambda_{\text{look}}})}{cost_{\text{split}}(C) + cost_{\text{opr}\uparrow}(C', R', \overline{R'}, d+1) + cost_{\text{opr}\downarrow}(C', R', \overline{R'}, d+1)} \tag{4}$$

be the objective function to be maximized subject to constraints

$$cost_{\text{opr}\uparrow}(\cdots) < B_{\text{C}\uparrow} \ , \tag{5}$$

$$cost_{\text{opr}\downarrow}(\cdots) < B_{\text{C}\downarrow} \ , \tag{6}$$

$$\lfloor C/2 \rfloor > C_{\min} \ , \tag{7}$$

$$\lambda_{\text{join}} - \lambda_{\text{leave}} > 0 \ . \tag{8}$$

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 (8), might become negative after a split. Values of $C_{\min}$ are unique for each group. With a method presented in [2], coordinators compute a prediction of $C_{\min}$ whenever an event in their groups occurs. The denominator of eq. (4) 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. If the ratio in (4) is maximal and all constraints hold, then this defines the best point when to split a group. Constraints (5) and (6) have special importance. If one or both of them become false and constraint (8) is true but constraint (7) 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.

5 Experiments for split threshold determination

We now briefly describe our simulator, the experiments that show the behavior of the optimization model, and that computed split thresholds are practical in various scenarios.

5.1 Simulation setup

We developed an activity-based simulator written in Java that allows concurrent, non-deterministic peer interactions. 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.

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. 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. For details about parameter settings, we have to refer the reader to [3] due to space constraints.

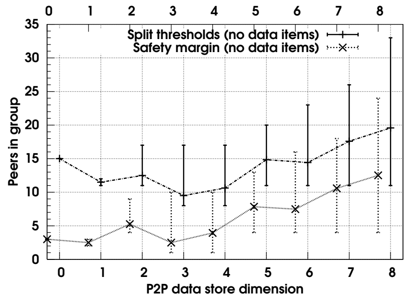

Figure 1. Simulation results varying number of groups

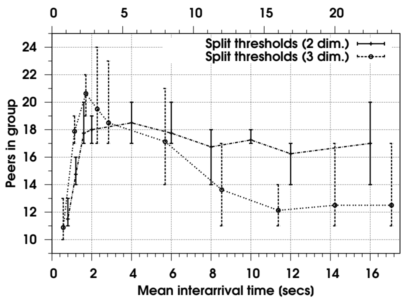

Figure 2. Simulation results varying interarrival time of turnover events

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 smallest, average, and greatest 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. The explanation for this is: the turnover rate halves from dimension x to dimension $x+1$ and is highest in dimension 0. We note that split thresholds in dimension one and two overshoot their optima because of the high turnover rate. The observed increase in average split thresholds in dimensions > 3 is due to operational costs of those groups growing faster than the sum of operational costs of a group resulting from a split plus the 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 a split threshold (first curve) and $C_{\min}$ and grow with the size of the P2P data store.

The effects of varying mean interarrival times of join and leave events on split thresholds are shown in Figure 2. For these experiments, we inserted 1000 data items overall and recorded split thresholds from groups in a 2- (lower x-axis) and 3-dimensional (upper x-axis) P2P data store. Error bars again express the minimum, average and maximum split thresholds of groups in the same dimension. We see a slow decrease of average split thresholds down to a lower bound for mean interarrival times > 2 secs. in the P2P data store with 3 dimensions and > 4 secs. in the 2-dimensional P2P data store. This happens because decreasing the turnover rate also decreases the operational costs. For mean interarrival times less than the above mentioned values, we see a sharp decrease of split thresholds. This surprising phenomenon has a simple explanation: A high turnover rate also leads to high $C_{\min}$ values because when sibling groups merge there must be enough peers in both groups to guarantee data availability during and after the fusion. A high turnover rate also increases coordinators' communication a lot, which in turn might exceed their maintenance bandwidth budget quickly. Both effects determine split thresholds (cf. constraints (5), (6), and (7)) when mean interarrival times are below 2 or 4 secs., respectively. Also notice, if the mean interarrival time of events in a group is shorter than 400 ms, no split will ever be performed. This is because constraint (7) never holds.

Notice, our technical report [3] has two additional series of experiments that show how varying number of data items and the lookup/turnover events ratio affect split thresholds.

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 used simulations to validate our model. The simulation results show that reasonable optimal split thresholds can be computed and there is always enough safety margin between split thresholds and minimum numbers of peers necessary for a coalescence of two groups. However notice, all maintenance operations ought to be invoked infrequently because of their bandwidth consumption.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research.* 5th ed., McGraw-Hill Inc., 1990
7. 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.
8. 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.
9. 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.
10. S. Marti and H. Garcia-Molina. Taxonomy of Trust: Categorizing P2P Reputation Systems. Computer Networks 50(4): 472–484, 2006.
11. S. Rhea, D. Geels, et al.. Handling Churn in a DHT. In *Proc. of the USENIX Annual Technical Conference*, Boston, MA, USA, 2004.