A Self-Organizing Job Scheduling Algorithm for a Distributed VDR

Kendy Kutzner, Curt Cramer, and Thomas Fuhrmann
IBDS Systemarchitektur, Universität Karlsruhe (TH)
{kutzner|cramer|thomas.fuhrmann}@ira.uka.de

1 Background

In [CKF04], we have reported on our concept of a peer-to-peer extension to the popular video disk recorder (VDR) [Sch04], the *Distributed Video Disk Recording (DVDR) system.* The DVDR is a collaboration system of existing video disk recorders via a peer to peer network. There, the VDRs communicate about the tasks to be done and distribute the recordings afterwards. In this paper, we report on lessons learnt during its implementation and explain the considerations leading to the design of a new job scheduling algorithm.

DVDR is an application which is based on a distributed hash table (DHT) employing proximity route selection (PRS)/proximity neighbor selection (PNS). For our implementation, we chose to use Chord [SMK$^+$01, GGG$^+$03]. Using a DHT with PRS/PNS yields two important features: (1) Each hashed key is routed to exactly one destination node within the system. (2) PRS/PNS forces messages originating in one region of the network destined to the same key to be routed through exactly one node in that region (*route convergence*). The first property enables per-key aggregation trees with a tree being rooted at the node which is responsible for the respective key. This node serves as a rendezvous point. The second property leads to locality (i.e., low latency) in this aggregation tree.

In our specification of the algorithm (cf. sec. 2), we distinguish inner nodes from leaf nodes. The latter both issue and fulfill recording requests ("jobs"). The former aggregate these requests. In an actual system, however, nodes are able to conduct both tasks. We model this by representing a physical node that happens to be an inner node of the aggregation tree by two virtual nodes, one inner (for aggregation) and one leaf node (for job creation and fulfillment).

Node churn will create inconsistencies in this structure, but these inconsistencies are temporary and limited to a local scope. If, e.g., the node that is the rendezvous point for a certain key unexpectedly dies, most parts of the aggregation tree for that key remain intact. (The tree lost its root, but aggregation in the sub-trees still works.) When a another node steps in as a new root, this information quickly propagates down the tree, causing a quick reorganization of its sub-trees. Hence, at time-scales well above the average DHT lookup latency (about 400ms in a globally deployed system [KF05]), the system appears to be stable. (Note that the very same argument discussed here for the rendezvous point also applies to the disappearance of any other node in the system.)

2 The Algorithm

Assumptions and Goals

We assume that all nodes in the system have reasonably synchronized clockswhich is easily achievable with standard protocols like the Network Time Protocol (NTP). Each job k is assigned a discrete starting time t_k, where t_k is rounded to the next full minute divisible by five. (This coarse slicing of the time is the reason for the lax synchronization requirements. It is the basis for the definition of timeout values and the system's ability to fuse overlapping data blocks stemming from different nodes.) Each job is identified by a hash key h obtained from hashing the job's starting time $h = H(t_k)$. A leaf node i issuing a job J_k also has to announce a priority $p_{i,k} \in \{1, 2, \ldots\}$ for it. There, $p = 1$ denotes a high-priority job (as rated by the respective node's user). A value of $p = 1$ is also assigned to jobs which can be especially well performed by a node (e. g. because its receiver card has already been tuned to the designated channel). A node may handle several jobs with the same starting time, however these jobs have to differ in their priorities. It is highly probable that several nodes will issue requests for the same job (not necessarily with differing priority assignments). The goal of the scheduling algorithm then is to identify a subset of all nodes that execute J_k. We assume that, for every possible t_k, there are fewer jobs than nodes in the system (i. e. the number of channels is smaller than the number of nodes). This is leveraged to have each job be redundantly executed on several nodes. The goal of our algorithm is that the higher the job's global priority $p_k = \left(\sum_i p_{i,k}^{-1}\right)^{-1}$ is, the more nodes should execute it[1]. The higher the job's local priority $p_{i,k}$ is, the more likely node i will perform the job itself.

Details

At point in time $t = t_k - T_1$, where T_1 is randomly drawn from [2.5 min,7.5 min], node i whose highest priority job is J_k issues a REQUEST message with request value $R_k = 1$ and a COMMIT message with commit value $C_k = 1$, i. e. the node requests some node to execute the job and commits itself to do it unless further notice is given. For all lower priority jobs, it issues a REQUEST message with $R_{k'} = p_{k'}^{-1}$, but a COMMIT message with $C_{k'} = 0$. The messages are destined for key $h = H(t_k)$. Since requests and commits always come in pairs, we will refer to these messages as RC messages. These original messages are only forwarded one overlay hop towards the destination key h, as messages always get aggregated (i. e. the amount of requests R and the number of commits C are summed up) on the next node. This is done as follows:

[1]Note that, as with $p_{i,k}$, the lower p_k is, the higher the priority is.

When a node receives an RC message from node i, it creates or updates a record containing $R_{i,k}$ and $C_{i,k}$. If this was the first RC message for that job, it also sets a timer T_2, where T_2 is randomly drawn from $[5\ s, 15\ s]$. (Note that T_1 should be much larger than T_2.) If further RC messages arrive for J_k, their respective parameters are recorded, too. Thus, the node builds a cumulated list of direct downstream[2] overlay neighbors for that job.

When the timer fires, the node issues an RC message containing the aggregated parameters $\sum_i R_{i,k}$ and $\sum_i C_{i,k}$ towards h (upstream) and sends a SUPPRESS message with $S_k = \log_2(1 + \sum_i R_{i,k}) / \sum C_{i,k}$ to all nodes in its list, i. e. SUPPRESS messages are relayed hop-by-hop downstream towards the leaves of the aggregation tree. Note that $S_k = \infty$ is a valid result. We chose a logarithmic relationship between requests and commitments as target function because requests coming from only a few nodes should be recorded by these nodes themselves. When a lot of nodes have similar requests, most of the nodes can perform lower priority tasks.

Upon reception of a SUPPRESS message, an *inner node* calculates the mean of both the received S_k^{recv} and the S_k^{calc} it calculates for its own current list of parameters. Then, it sends a SUPPRESS message containing this mean value towards its downstream neighbors and resets its timer T_2.

Upon reception of a SUPPRESS message, a *leaf node* draws a uniform random number $r \in [0; 1[$. If $r < S_k$, the node sticks to its current commitment to perform J_k and again issues a corresponding RC message. Otherwise, it switches to its next lower priority job $J_{k'}$ and sends two RC messages, one revoking its commitment for J_k and one committing it to $J_{k'}$. If a leaf node has no further lower priority jobs to be performed, it commits to the job with the largest S_k value.

This algorithm has three important features: (1) Calculating the mean value leads to subtrees having suppression values that more and more reflect the local requests rather than the global view at the root of the aggregation tree. (2) Over time, S_k values tend to become equal, i. e. request and commit values are (from a local perspective) in a fair balance. (3) All inner nodes are functionally equal. There is no special role assumed by the root. Hence, arbitrary subtrees may be pruned from the main tree at any time without affecting the functionality of both these subtrees and the rest of the system.

Damping

The algorithm as described above can cause oscillations. In a system where there are more jobs than nodes, the oscillations would eventually die off (the larger the gap between requests and commitments, the quicker they die off). In DVDR, the opposite scenario prevails. Therefore, we have to add an additional damping mechanism. It works as follows:

1. Nodes committed to a job with priority p_k will only change their commitment after time $t_k - T_1(1 - \alpha p_k^{-1})$ (where α is randomly drawn from $[0, 1]$), i. e. the higher the priority of a request is, the more reluctant a node is to rely on another node to do the job.

[2]Downstream designates the direction from the aggregation tree's root towards its leaf nodes.

2. Upon reception of a SUPPRESS message, a node calculates $m = \max\{S_{k_1}, \ldots, S_{k_n}\}$ and draws a uniformly distributed random number $r \in [0; m[$. Let J_{k_1} be the job to which the node committed most recently. If $r < S_{k_1}$ (which is very likely when the S_k-values become similar), the node sticks to its commitment. Otherwise, it (uniformly distributedly) chooses a job from the set $\{J_k | r < S_k\}$ and commits to doing that job.

3 Conclusions and Outlook

We presented a fully distributed scheduling algorithm for video disk recorder schedules. Our algorithm leverages the overlay structure that is created by PNS/PRS in DHTs to reflect locality and ensure requests being preferably fulfilled close to their origin. Since our specific application system can be assumed to be highly redundant (many more active recorders than channels to record), our algorithm does not need to find an optimum solution. It suffices to create a fair schedule in a fully decentralized way. Currently, we are optimizing and implementing the described algorithm in the DVDR. One optimization is to further minimize the message count, since for each possible recording time there is only one aggregation tree. This tree can be used to schedule all channels at once. We hope to soon be able to report on the achievable performance.

We believe that this algorithm does not only provide an elegant solution to a practical problem in our P2P application, but also sheds some light onto the potential that P2P approaches have beyond mere file sharing applications.

References

[CKF04] Curt Cramer, Kendy Kutzner und Thomas Fuhrmann. Distributed Job Scheduling in a Peer-to-Peer Video Recording System. In *Proceedings of the Workshop on Algorithms and Protocols for Efficient Peer-to-Peer Applications (PEPPA) at Informatik 2004*, Seiten 234–238, Ulm, Germany, September 23 2004.

[GGG$^+$03] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker und I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proceedings of the SIGCOMM 2003 conference*, Seiten 381–394. ACM Press, 2003.

[KF05] Kendy Kutzner und Thomas Fuhrmann. Measuring Large Overlay Networks - The Overnet Example. In *14. Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, 2005.

[Mil92] D. Mills. Network Time Protocol (Version 3) Specification, Implementation. RFC 1305 (Draft Standard), Marz 1992.

[Sch04] Klaus Schmidinger. Video Disk Recorder, 2004. http://www.cadsoft.de/vdr/, accessed on 12 May 2004.

[SMK$^+$01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek und Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the SIGCOMM 2001 conference*, Seiten 149–160. ACM Press, 2001.