Coded-Seeking: A Simple HDD Speed-Up Concept
Ulric J. Ferner and Muriel Médard

Abstract—Consider a single hard disk drive (HDD) composed of rotating platters and a single magnetic head. We propose a simple internal coding framework for HDDs that uses coding across drive blocks to reduce average block seek times. In particular, instead of the HDD controller seeking individual blocks, the drive performs coded-seeking: It seeks the closest subset of coded blocks, where a coded block contains partial information from multiple uncoded blocks. Coded-seeking is a tool that relaxes the scheduling of a full traveling salesman problem (TSP) on an HDD into a k-TSP. This may provide opportunities for new scheduling algorithms and to reduce average read times.

Index Terms—Coded-seeking, disk drives, HDD, network coding, read-ahead, scheduling, speed-up.

I. A RATIONALE FOR CODED-SEEKING

HARD DISK drives, composed of mechanical arms and platters, have been a staple of storage networks for decades. Not only has the cost of HDDs steadily declined, but also the density of drive data has increased in tandem, yielding ever-cheaper and more effective storage technology [1]; this has secured their use in systems like content distribution [2]. Despite these trends, solid state drives have become increasingly popular owing to performance benefits, especially in portable devices. Their lack of moving mechanical parts means that even when data is randomly distributed on a solid state drive the read times are near constant and there is no physical read-head bottleneck [1]. In contrast, HDD physical movement is often a bottleneck in modern storage systems as their read times for a single block can be on the order of a few ms to tens of ms [3].

We divide HDD block read times into seek and block load times, where seek time is the time taken for the platters and a head to physically move into position over the relevant block, and block load time is the remaining time taken to read the block after the head has been positioned over the block. The physical movement required for HDDs seeking has motivated numerous read-ahead algorithms to reduce input/output (I/O) latency in HDDs [4]–[6]. Generally, these operate as follows. Given a set of received block requests, controllers can use read-ahead algorithms to predict future access and preemptively read blocks ahead of received requests. We refer to [7] for a general overview of existing I/O algorithms and to [3], [8], [9] for treatment of select read-ahead algorithms.

The use of coding has been used in HDDs for some time for error correction within single blocks. This includes the use of Reed-Solomon codes and, more recently, has been extended to LDPC codes [10]. However, to the authors’ knowledge, coding has not been used to reduce I/O latency in HDDs. Nonetheless, given a set of received block requests, some existing algorithms read block requests out of order to reduce physical drive movement, such as shortest access time first (SATF) scheduling [7]. For instance, a controller could seek the closest one-of-many data-blocks that contain useful information, as opposed to simply seeking the next block request in the queue.

Given a group of blocks on an HDD that store a single file, at the cost of additional storage space, copying and storing additional or redundant copies of some blocks obviously shrinks the average physical distance between blocks and perhaps the associated average seek time for the entire group. In this paper, we propose a coded-seeking framework that may reduce seek times of groups of blocks: this framework allows drives to store additional coded blocks, as opposed to a selection of uncoded blocks, to increase scheduling flexibility and amplify potential seeking savings. Alternately, given a drive with fixed file sizes, this framework gives drive designers the option to store less information per fixed-size file, and thereby increase scheduling flexibility and amplify seeking savings of these reduced files.

In particular, a set of \( k \) uncoded blocks are coded together into \( n \geq k \) coded blocks using a maximum distance separable (MDS) code, such that any \( k \) out of the \( n \) coded blocks allow decoding. The coding parameter \( n \) relaxes reading scheduling constraints, which can reduce average seek times, and to our knowledge, such flexibility is more challenging to guarantee using uncoded systems.

II. SYSTEM MODEL

Consider a single HDD composed of platters and a single read-head.\(^1\) The platters are divided into equally-sized blocks. Each block contains a small amount of data that can be read out from the HDD: for instance, in the common Advanced Format Standard, blocks are of size 4096 bytes [11], [12]. The control interface is as follows. The HDD is equipped with a drive controller that receives block requests from the operating system; it has access to an HDD cache connected to the HDD platters through an I/O bus [13]. Suppose the HDD contains a set of file blocks \( F = \{ f_i \}_{i=1}^k \), where \( f_i \) is the \( i \)th individual file block.\(^2\) Read requests arrive at the controller for \( F \), and blocks must be located on and read from the platters into a drive cache. We assume blocks can be read out of the cache

\(^1\)It is straightforward to generalize the results in this paper to drives with multiple heads.

\(^2\)As defined by the operating system, \( F \) may be a complete logical file, greater than one logical file, or a subset of a single logical file.
in a way we model as instantaneous [14]. Our metric for this system is drive seek time, which is the average time required for \( \mathcal{F} \) to be read and decoded from a disk when a single request arrives. Generally in the worst case for seek time, blocks may be distributed around disk cylinders on the platter randomly in a uniform distribution in two-dimensions [15], [16].

III. CODED-SEEKING

We now describe the coded-seeking framework and explore seek time implications.

A. Design

As opposed to storing a set of \( k \) uncoded file blocks \( \mathcal{F} = \{f_1, f_2, \ldots, f_k\} \), we instead store a set of coded file blocks \( \mathcal{C} = \{c_1, c_2, \ldots, c_n\} \), where \( n \geq k \). Note that the total storage consumed by \( \mathcal{C} \) can be greater than or equal to that consumed by \( \mathcal{F} \), although integer multiples of content size are not required. The set of coded blocks \( \mathcal{C} \) is constructed as a maximum distance separable (MDS) block code \((n,k)\), which is composed of \( n \) coded blocks, such that any \( k \) coded blocks can be used to decode \( \mathcal{F} \) [17]. A Reed-Solomon code is an example of an MDS code. Blocks, be they coded or not, consume the same storage space, and we assume that coding coefficients are stored similarly to coefficients of Reed-Solomon codes for block error correction, in the meta-data portion for each block [10].

In this framework, coded block \( c_i \) contains partial information on all blocks in \( \mathcal{F} \). Similarly to [18] and [19], when a coded block is read from the platters into the drive cache, both \( c_i \) as well as any corresponding meta-data is also read into the cache. The operating system or the drive cache decodes \( \mathcal{F} \) from a subset of \( \mathcal{C} \), when \( k \) unique coded chunks have been read into the cache. See Fig. 1 for a simple illustration.

B. Performance Formulation

We illustrate the performance of coded-seeking by formulating the seeking of these \( k \) coded blocks as a graph theoretic problem, assuming the cache is empty at the time of a read arrival for \( \mathcal{F} \). The formulation will show that coded-seeking transforms a TSP on an HDDs into a \( k \)-TSP.

Consider a weighted directed graph \( G = (V, E, W) \), where \( V = \{s, 1, \ldots, n\} \) is a set of \( n+1 \) vertices representing the physical locations of \( \{c_i\}_{i=1}^n \) on the platters plus an additional starting vertex \( s \) that represents the starting location of the read head; \( E \subseteq V \times V \) is a set of edges; and \( W = \{w_{i,j}\}_{(i,j) \in E} \) is a set of nonzero weights, where \( w_{i,j} \) represents the head seeking time to move from vertex \( i \)'s physical position to vertex \( j \)'s physical position, as illustrated in Fig. 1. We set \( w_{i,i} = \infty \), \( \forall i \in V \).

Definition: The coded-seeking problem is to find the minimum weight trail [20] over \( G \) that traverses at least \( k+1 \) unique vertices in \( G \), which includes the starting vertex \( s \). No vertex may be visited more than once.\(^3\) Denote the subgraph of \( G \) that forms the minimum weight trail as \( G_{\text{seek}} = (V_{\text{seek}}, E_{\text{seek}}, W_{\text{seek}}) \subseteq G \), and define \( x_{i,j} \)'s as binary variables:

\[
x_{i,j} = \begin{cases} 
1 & \text{if } (i,j) \in E_{\text{seek}}, \\
0 & \text{otherwise}.
\end{cases}
\]

We reformulate the coded-seeking problem as an optimization problem as follows. Construct weighted and directed graph \( G_{\text{opt}} = (V_{\text{opt}}, E_{\text{opt}}, W_{\text{opt}}) \) such that \( V_{\text{opt}} = V \cup \{d\} \), where \( d \) is a dummy destination vertex. Let \( E_d = \bigcup_{i \in V_{\text{opt}}} (i,d) \), and let all edges in \( E_d \) have weight zero, and define \( E_{\text{opt}} = E \cup E_d \). The coded-seeking problem can now be formulated as an integer linear program (ILP) formulation very similar to a shortest path problem from \( s \) to \( d \) or a traveling salesman problem without return, where a path traverses at least \( k \) unique vertices representing coded blocks, in addition to start node \( s \in V_{\text{opt}} \) and end node \( d \in V_{\text{opt}} \). We use the following ILP formulation, similar to those in [21],

\[
L_{n,k} = \min \sum_{(i,j) \in E_{\text{opt}}} x_{i,j}w_{i,j}
\tag{1}
\]

such that

\[
\sum_{j \in V_{\text{opt}}, j \neq i} x_{i,j} - \sum_{j \in V_{\text{opt}}, j \neq i} x_{j,i} = \begin{cases} 
1 & i = s \\
-1 & i = d \\
0 & \text{else}
\end{cases}
\tag{2}
\]

\[
\sum_{(i,j) \in E_{\text{opt}}} x_{i,j} \geq k + 1
\tag{3}
\]

\[
\sum_{j \in V_{\text{opt}}} x_{i,j} \leq 1 \quad i \in V_{\text{opt}}
\tag{4}
\]

\[
x_{i,j} \leq 1 \quad (i,j) \in E_{\text{opt}}
\tag{5}
\]

\[
x_{i,j} \in \mathbb{Z} \quad (i,j) \in E_{\text{opt}}
\tag{6}
\]

\[
u_i - u_j + (k+2)x_{i,j} \leq k + 1 \quad i,j \in \{1, \ldots, n\}
\tag{7}
\]

\[
u_i \geq 0 \quad i \in \{1, \ldots, n\}
\tag{8}
\]

\(^3\)A trail is defined as a walk of vertices and edges with no repeated edge. A walk is a list \( v_0, e_1, v_1, \ldots, e_m, v_m \) of vertices and edges such that, for \( 1 \leq i \leq m \), the edge \( e_i \) has endpoints \( v_{i-1} \) and \( v_i \). The restriction that no vertex is visited more than once can be removed if necessary. It is introduced here only to simplify the problem.
where \( \{ u_i \} \) is a set of artificial variables, such that \( u_i \) is associated with vertex \( i \). The artificial variable constraints restrict the feasible solution space to that of closed paths. The solution is graph \( G_{\text{seek}} = G \cap G_{\text{opt}} \) with the same cost function value as (1).

**Theorem 1:** Every solution to \( L_{n,k} \) is a single trail and all single trails of sufficient length are feasible.

**Proof (Sketch):** This proof is similar to proofs for the traveling salesmen problem found in [21]. First, we show that every feasible solution is a trail. Assume vertices \( i_1, \ldots, i_m \) form a sub-trail that does not include \( s \), the head starting position. Constraint (2) implies it must be a closed trail because equal in and out flow is required for all intermediate nodes. The constraints also imply that the closed trail has a single loop, owing to (4). We then have the following

\[
\begin{align*}
  u_{i_1} - u_{i_2} + (k + 2)x_{i_1,i_2} &\leq k + 1 \\
  u_{i_2} - u_{i_3} + (k + 2)x_{i_2,i_3} &\leq k + 1 \\
    &\vdots \\
  u_{i_m} - u_{i_1} + (k + 2)x_{i_m,i_1} &\leq k + 1.
\end{align*}
\]

Summing the constraints yields \( m(k + 2) \leq m(k + 1) \), which is a contradiction. Hence there are no closed trails that do not include \( s \), so every feasible solution is a trail.

Now we show that every valid trail is feasible, i.e., we show that every trail that satisfies all constraints (2)–(6), can also satisfy (7), (8). Let \( i_1, \ldots, i_m \) form a trail. Without loss of generality, we momentarily define artificial variables \( \{ u_i \} \) by labeling vertices such that \( i_1 = 1, i_2 = 2, \ldots, i_m = m \) and

\[
  u_{i_j} = \begin{cases} 
    j & i_j \in \{1, \ldots, k + 1\} \\
    k + 1 & i_j \in \{k + 2, \ldots, n\}.
  \end{cases}
\]

If \( x_{i_j,i_l} = 1 \), then \( i_j \) and \( i_l \) are consecutive vertices in the trail. This means \( l = j + 1 \) so

\[
  u_{i_j} - u_{i_l} = 1 - 1 = -1,
\]

implies \( u_{i_j} - u_{i_l} + k + 2 \leq k + 1 \) is satisfied.

If \( x_{i_j,i_l} = 0 \), then

\[
  u_{i_j} - u_{i_l} \leq k + 1
\]

must also be satisfied. In this case \( (i_j, i_l) \) are not consecutive nodes in the trail. If \( (i_j, i_l) \) are not consecutive and both members of \( V_{\text{seek}} \), then we have three cases to consider with respect to these vertices and the set \( V_{\text{seek}} \). First, consider \( i_j, i_k \notin V_{\text{seek}} \).

Eq. (9) sets \( u_{i_j} = u_{i_k} = k + 1 \) so (10) is satisfied. Second, consider \( i_j \in V_{\text{seek}}, i_l \notin V_{\text{seek}} \). This sets \( u_{i_j} = j, u_{i_l} = k + 1 \), so (10) holds. Third, consider \( i_j \notin V_{\text{seek}}, i_l \in V_{\text{seek}} \). This sets \( u_{i_j} = k + 1, u_{i_l} = l \) so \( u_{i_j} - u_{i_l} = k - l + 1 \) and since \( l \geq 1 \), then \( k - l + 1 \leq k + 1 \). This implies every valid trail is feasible.

As per (1), current HDDs, without coded-seeking, solve \( L_{k,k} \) or \( L_{n,n} \) which is a full TSP. Coded-seeking is a tool that allows us to transform \( L_{k,k} \) or \( L_{n,n} \) into \( L_{n,k} \), where

\[ n > k. \]

Solving \( L_{n,k} \) is an application of the well studied \( k\text{-TSP} \) problem, which is known to be \( NP \)-hard, even in Euclidean \( \mathbb{R}^2 \) space [22]. This means that implementing exact solutions to the coded-seeking problem in drive controllers is unlikely. Instead, approximations or heuristics can be used instead, similar to those already used to help with TSPs already encountered in fragmented HDD. Encouragingly, as with TSPs [23], there exist \( k\text{-TSP} \) approximation algorithms which achieve constant-factor approximations in \( \mathbb{R}^2 \) [24]. However, prior to considering coded-seeking approximations, we explore potential gains given a fragmented HDD and the ability to solve \( L_{n,k} \), as per (1), exactly.

As discussed previously, coded-seeking allows designers to increase scheduling flexibility by either (1) storing additional coded blocks, by fixing \( k \) and increasing \( n \); or (2) given fixed flat-file sizes of \( n \) blocks, reducing \( k \). To illustrate potential benefits, owing to space limitations, this paper considers only the latter. Specifically, Fig. 2 considers \( k \) block coordinates uniformly distributed across the unit circle—depicting a circular platter—and solves the coded-seeking problem \( L_{n,k} \) and the full TSP without return, \( L_{n,n} \), where the weight between any two vertices is their Euclidean distance.\(^4\) We normalize numerical results by \( L_{n,n} \) and plot \( L_{n,k}/L_{n,n} \) as a function of \( k/n \); each point is averaged across fifty random networks. Fig. 2 also plots the line \( k/n = L_{n,k}/L_{n,n} \) for reference. We make two remarks. First, the results are stable across \( n \), from

\[ n = 5, 10, 20 \]

\[ \text{Ref.: } x = y \]

\( ^4 \) A number of related functions could also be compared to \( L_{n,k} \), such as \( L_{k,k} \) or uncoded-seeking results with particular replication strategies and particular block layout probabilities. We do not show such results here owing to space limitations, and instead focus on the simple scaling of \( L_{n,k} \), where \( k \) is a fraction of \( n \).
5 up to 20. Second, and most importantly, as the fraction of coded blocks required to decode, $k/n$, decreases, then the fraction of required seek distance, $L_{n,k}/L_{n,n}$, decreases at faster rate. For instance, if we traverse 50% of nodes ($k/n = 1/2$), then $L_{n,k}/L_{n,n}$ is less than 50%, so we see a disproportionate gain in path length.

Contrasting illustrations with small $k$ and $n$, it has been also shown in $k$-TSPs that, as $n \to \infty$ and given particular assumptions, asymptotically the expected path length in random graphs scales approximately linearly as a function of $k/n$. Also, if $k/n$ stays constant, then path length scales sub-linearly [25].

IV. COMMENTS ON CODED WRITING

There may be interesting opportunities for coded-seeking to reduce HDD read times, although these opportunities bring with them numerous complications and open questions. A key open question is the converse to coded-seeking: coded-writing. The writing of data using combinations of blocks is more complex and a slower process compared to uncoded writing. This may make coded-seeking useful in only select applications with particular read and write patterns, and with particular $(n, k)$ requirements.

The benefits of coded-seeking are likely highest when read requests require the HDD to access blocks uniformly and randomly on a platter. When requests are predictable, then the seek time reductions from coded-seeking may only be marginal and may ultimately be outweighed by complexities from coded writing. Again, data and number-of-user patterns will be key in understanding the full implications of this concept.

V. CONCLUSION

Modern HDDs are often a bottleneck in read-heavy content distribution systems. Nonetheless, their cost benefits over solid-state drives will likely motivate their continued use in practice for some time. Coded-seeking is a simple technique to reduce seek times, in which blocks are coded together. The promise of this technique will likely require the careful design of coded writing as well as an understanding of likely read or write patterns in applications of interest.

ACKNOWLEDGMENT

The authors would like to acknowledge Prof. David Karger for his helpful discussions and the reviewers for their careful reading of the manuscript.

REFERENCES