object offsets to file offsets and issues a `TIPIO_SEG` hint. Such disclosure is consistent with the module interfaces already in place; the DFSD layer issues hints about data objects and the `Hhint` routine translates these data-object hints into file-access hints which it discloses directly to TIP. The modularity of the HDF library is not a barrier to hints that disclose.

The `Hhint` routine provides the DFSD layer with a modular mechanism for issuing hints, but the DFSD layer still needs to find a way to call `Hhint` to issue hints. We accomplish this through loop splitting.

The DFSD layer receives a list of the coordinates of the submatrices required by the XDataSlice application to render a slice. The original, unhinting code loops over these coordinates translating each to an offset within the data object and then calling `Hseek` followed by `Hread` to retrieve the needed block. Inserting an `Hhint` call within this loop would not provide much advance warning. An alternative is to duplicate the loop and translate the coordinates and issue hints. The unchanged, original loop would re-translate the coordinates and perform the seek and read. To save the cost of re-translating the coordinates, I take advantage of the fact that the translated coordinates are passed in an array to `Hhint`. I use this same array to store the translated coordinates from the first loop and iterate over these in a second loop. Thus, the original loop is split to deliver hints efficiently.

3.4.6 Sphinx

Sphinx [Lee90] is a high-quality, speaker-independent, continuous-voice, speech-recognition system developed at Carnegie Mellon. In the benchmark, Sphinx recognizes an 18-second recording commonly used in Sphinx regression testing.

Sphinx represents acoustics with Hidden Markov Models and uses a Viterbi beam search to prune unpromising word combinations from these models. To achieve higher accuracy, Sphinx uses a language model to effect a second level of pruning. The language model is a table of the conditional probability of word-pairs and word-triples. At the end of each 10 msec acoustical frame, the second-level pruner is presented with the words likely to have ended in that frame. For each of these potential words, the probability of it being recognized is conditioned by the probability of it occurring in a triple with the two most recently recognized words, or occurring in a pair with the most recently recognized

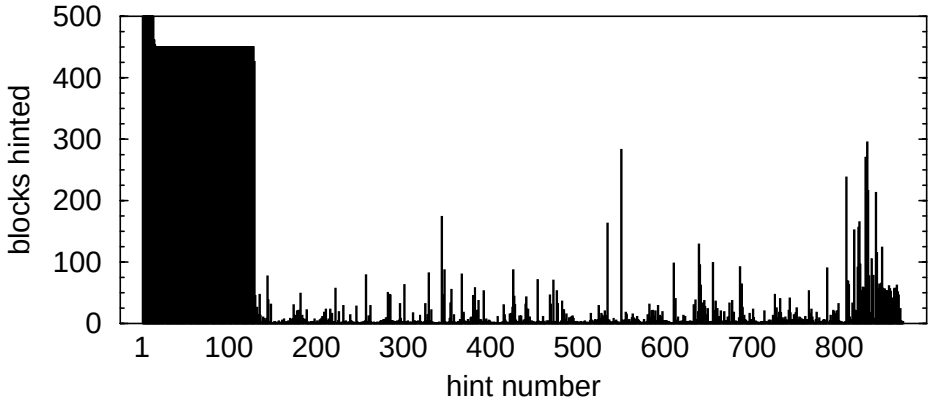

Figure 3.6. Sphinx: blocks hinted in each hint. This graph shows the distribution of the number of blocks hinted by each of Sphinx's 873 hints. The first approximately 120 hints are for initialization and disclose a maximum of 2477 blocks. The rest disclose dynamic loads of language model data as needed during the course of recognizing the speech segment. The average hint during the recognition phase is for about 15 blocks, although many are for a lot more than that.

word when there is no entry in the language model for the current triple. To further improve accuracy, Sphinx makes three similar passes through the search data structure, each time restricting the language model based on the results of the previous pass.

Sphinx, like XDS, was originally an in-core only system. Because it was commonly used with a dictionary containing 60,000 words, the language model was several hundred megabytes in size. With the addition of its internal caches and search data structures, virtual-memory paging occurs even on a machine with 512-MBytes of memory. Sphinx was modified to fetch the language model's word-pairs and word-triples from disk as needed[9]. This enables Sphinx to run on a 128-MByte test machine 90% as fast as on a 512-MByte machine.

Sphinx was also modified to disclose the word-pairs and word-triples that will be needed to evaluate each of the potential words offered at the end of each frame. Figure 3.6 shows the distribution of the number of blocks hinted by each of Sphinx's 873 hints. The hints for the initialization-phase reads disclose a high degree of concurrency. Hints during the recognition phase are highly variable. Because the language model is sparsely populated, at the end of each frame there are about 100 byte ranges that must be consulted, of which all but a few are in Sphinx's internal cache. However, there is a high variance on

[9] Thanks to Daniel Stodolsky who modified Sphinx to operate out-of-core and then annotated it with hints.

the number of pairs and triples consulted and fetched, so, although the hints provide little advance warning, they often expose I/O concurrency.

3.5 Conclusion

The first hurdle in making the case that application hints about future file reads can compensate for the growing disparity between processor and disk performance is demonstrating that important applications can in fact give hints. Without this, there is no need to pursue this line of research any further.

In this chapter, I first argued for hints that disclose future accesses in preference to hints that give advice about filesytem behavior. The distinguishing feature of disclosure hints is that they are expressed using the same terms of file, byte offset, and byte length as the existing file-system interface. I presented a prototype interface for delivering disclosure hints.

I then described three techniques for annotating applications to give disclosure hints: in-line hinting, loop duplication, and loop splitting. In six application case studies, I described how to use the three techniques to annotate important, I/O-intensive applications to give hints. In one, XDataSlice, I showed how disclosure hints may be translated by intermediate software layers to preserve modularity.

How successful were the annotations at disclosing these application's read accesses? Table 3.2 reports the I/O workloads of the benchmarks and the percentage of the read traffic disclosed in advance by hints. For four of the applications, hints disclose more than 99% of the bytes read. Hints disclose 90% of the bytes read by a fifth application, Sphinx. Although hints disclose Postgres' random, data-dependent inner-relation accesses, no annotations disclose the large number of index accesses. However, as will be seen in Chapter 6 which describes the performance of the benchmarks, these index accesses cache well even without hints, and so, although incomplete, the hints yield huge performance wins for the application.

Hints may disclose every access, but if they don't also expose concurrency, then they don't add the parallelism to the read workload needed to relieve the I/O bottleneck. Fortunately, even though the applications are drawn from a broad range of fields, they all tend to give hints in bursts. Here, I define a burst as a sequence of hints given between hinted