Challenges for Dataset Search

David Maier, V.M. Megler, Kristin Tufte

Computer Science Department, Portland State University
{ maier@cs.pdx.edu, vmegler@cs.pdx.edu, tufte@cs.pdx.edu}

Abstract. Ranked search of datasets has emerged as a need as shared scientific archives grow in size and variety. Our own investigations have shown that IR-style, feature-based relevance scoring can be an effective tool for data discovery in scientific archives. However, maintaining interactive response times as archives scale will be a challenge. We report here on our exploration of performance techniques for Data Near Here, a dataset search service. We present a sample of results evaluating *filter-restart* techniques in our system, including two variations, *adaptive relaxation* and *contraction*. We then outline further directions for research in this domain.

Keywords: data discovery, querying scientific data, ranked search

1 Introduction

In the past, most scientific data was "one touch": it was collected for a specific experiment or project, analyzed by one or a small number of investigators, then used no further. The advent of shared scientific repositories – where data is collected prospectively and used by multiple scientists – is bringing an era of "high-touch" data. Each observation is now used tens or hundreds of times. A premier example of high-touch data is the Sloan Digital Sky Survey (SDSS), where a systematic imaging of the night sky has produced more than 5000 publications.[1] Such shared repositories not only provide higher returns on the investment in data collection, they enable new scientific discoveries, because of the broader spatial and temporal coverage. However, simply providing a shared data does not guarantee high "touchability" on its own. In a repository such as SDSS, with a handful of kinds of data coming from a common instrument, it is relatively easy for a scientist to determine what data is available that might match his or her needs. But in other domains, such as environmental observation, the number and variety of datasets grows over time, as new sensors and platforms are developed and deployed more widely. The situation can reach the point where no one person knows the full extent of the holdings of the repository. Scientists have a harder time finding relevant datasets for their work, and there is a true danger that increased growth in a repository actually decreases the touchability of data.

While there is a wide array of software tools for analysis and visualization of scientific data, there are fewer systems supporting data discovery. We heard this complaint

[1] http://www.sdss3.org/science/publications.php

adfa, p. 1, 2011.

from colleagues and visitors at the NSF-sponsored Center for Coastal Margin Observation and Prediction (CMOP). While CMOP provides interfaces, similar to those in other archives, to navigate its data holdings, their effective use requires prior knowledge of the kinds of data and their organization. Furthermore, those interfaces do not support finding datasets with particularly properties, such as "high salinity and low pH." Inspired by their needs, we have been investigating search tools to help locate datasets relevant to their information needs. We have adopted an approach to ranked search of datasets based on traditional Information Retrieval architectures, such as are used in web search. We prototyped our ideas in a service called Data Near Here (DNH), which we have deployed over the CMOP observation archive. Most searches execute quickly, and initial reaction is good, based on user studies with scientists, but we are concerned with scaling our approach as the archive expands.

We report here on some of the performance techniques we have examined, including a sampling of our evaluation results. We find that the *filter-restart* technique from top-k query processing can be effective in many cases. We have obtained further improvements by adjusting the cutoff threshold based on partial search results. The adjustment can be both downward between stages (*adaptive relaxation*) and upward within a stage (*contraction*). This work suggests many other avenues of investigation for scaling dataset search, some of which we describe in our final section.

2 Description of Data Near Here

At the high level, our architecture follows the search architecture used by most Internet search engines today. The adaptation to dataset search is shown in Fig. 1. The architecture consists of two sections, Asynchronous Indexing and Interactive Search, each of which communicates with a shared Metadata Catalog. Asynchronous Indexing consists of the *Crawl*, *Read* and *Extract Features* processes. The *Crawl* process identifies datasets to summarize. The *Read* process takes each dataset identified by the crawl process and attempts to read it. The *Extract Features* process asynchronously summarizes each read dataset into a set of features and stores the summary in a metadata catalog. The feature extraction process encapsulates dataset partitioning, hierarchy creation and feature extraction functions.

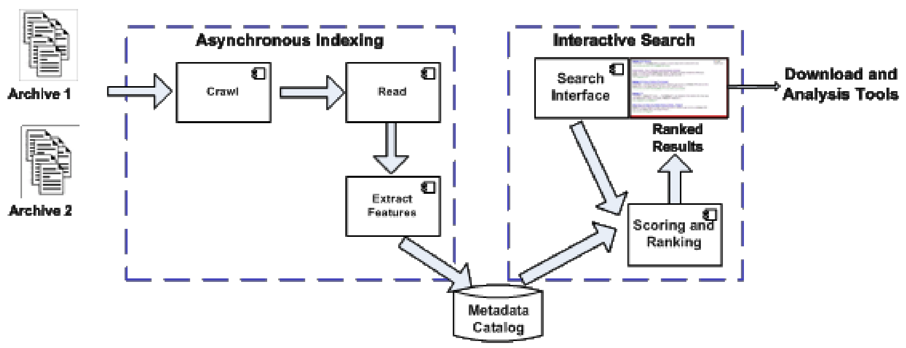


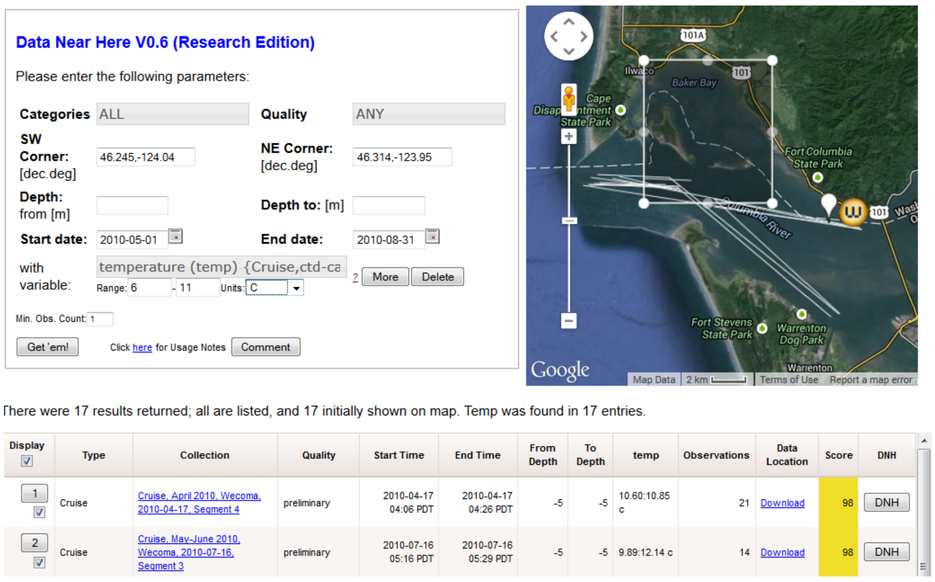
Fig. 1. High-level dataset search architecture

The primary processes in Interactive Search are the *Search Interface* and the *Scoring-and-Ranking* task. Searchers use the *Search Interface* to specify their search criteria. The Search Interface passes the search criteria to the Scoring-and-Ranking component, and displays the results returned. *Scoring-and-Ranking* accesses the metadata catalog, identifies the highest-ranked summaries for the given search criteria, and returns them to the Search Interface.

The *Metadata Catalog* stores the features and dataset summaries resulting from the asynchronous indexing and makes them available for use by other processes, such as the Scoring-and-Ranking process.

Our prototype implementation, Data Near Here [14, 15], has now been deployed for use by our scientists for over six months. The dataset summaries in DNH generally consist of the bounds of each variable, plus some file-level metadata. The one exception currently is latitude and longitude, which are jointly summarized as a geospatial footprint that is a geometry such as point, polyline or polygon. Search queries are similar to dataset summaries. DNH scores datasets against queries using a similarity function that considers distance and overlap between a dataset and the query, scaling different terms by the query extent. For example, if the query specifies a range of one day, then the previous and following day are considered a close match, whereas if the query specifies a month, the previous and following month are considered close.

Fig. 2 shows the primary user interface, which includes on a single webpage the search interface and ranked search results. The search interface is akin to the "advanced search" used by some text search engines, but adapted to scientific data. The



Data Near Here V0.6 (Research Edition)

Please enter the following parameters:

Categories	ALL	**Quality**	ANY
SW Corner: [dec.deg]	46.245,-124.04	**NE Corner:** [dec.deg]	46.314,-123.95
Depth: from [m]		**Depth to:** [m]	
Start date:	2010-05-01	**End date:**	2010-08-31
with variable:	temperature (temp) {Cruise,ctd-ca; Range: 6 - 11 Units: C		

Min. Obs. Count: 1

There were 17 results returned; all are listed, and 17 initially shown on map. Temp was found in 17 entries.

Display ✓	Type	Collection	Quality	Start Time	End Time	From Depth	To Depth	temp	Observations	Data Location	Score	DNH
1 ✓	Cruise	Cruise, April 2010, Wecoma, 2010-04-17, Segment 4	preliminary	2010-04-17 04:06 PDT	2010-04-17 04:26 PDT	-5	-5	10.60:10.85 c	21	Download	98	DNH
2 ✓	Cruise	Cruise, May-June 2010, Wecoma, 2010-07-16, Segment 3	preliminary	2010-07-16 05:16 PDT	2010-07-16 05:29 PDT	-5	-5	9.89:12.14 c	14	Download	98	DNH

Fig. 2. Search interface for "Data Near Here", showing a sample search for a geographic region (shown as a rectangle on the map) and date range, with temperature data in the range 6:11C. Result datasets (or subsets) are shown by markers and lines in the output pane, together with their relationship to the search region. In the ranked list of answers, no full matches for the search conditions were found; two partial matches to a search with time, space and a variable with limits are shown in the text panel, and the top 25 are shown on the map.

search results are displayed as brief summaries of the datasets, akin to the “snippets” of a document shown in web search. Details of a dataset’s metadata and contents can be viewed by clicking on an individual snippet. Because geospatial location is so important to our user base, the spatial extent of the search is shown on the map interface (the square white box), and the spatial extent of the datasets in the search results list are also shown on the map (here, diagonal lines and markers). However, our work is not limited to this kind of geospatial-temporal search.

One issue for scientists in finding relevant data is the mismatch between the scales of the data they seek, the scales of observation, and the partitioning of data for processing and storage. Multiple scientists might use the same datasets, but have very different scales of data of interest. Fig. 3 shows a dataset containing from a science cruise; the overall dataset is split into smaller segments representing specific sections of the cruise. Lynda, a microbiologist, wants data for a fairly short time period, since a different time in the tidal cycle might not reflect her sampling conditions. In Fig. 3, the most interesting portion of data for her is the cruise segment from July 28, 10-12 a.m., since it is closest in time and space to her time and location of interest. Another part of the cruise track passes her sampling location, but it is not close enough in time to be very relevant. Joel, an oceanographer, is looking for simultaneous low oxygen and lower pH; for him, the most relevant data is for the whole day of August 1 – a much larger portion of the dataset. Our metadata catalog supports such dataset hierarchies, and our search interface is aware of them. It considers datasets at all hierarchy levels for the results list, and may sometimes return both parent and child datasets. For example, it might return a single leg of a cruise as highly relevant to a query, while the full cruise might appear later in the list with lower overall relevance.

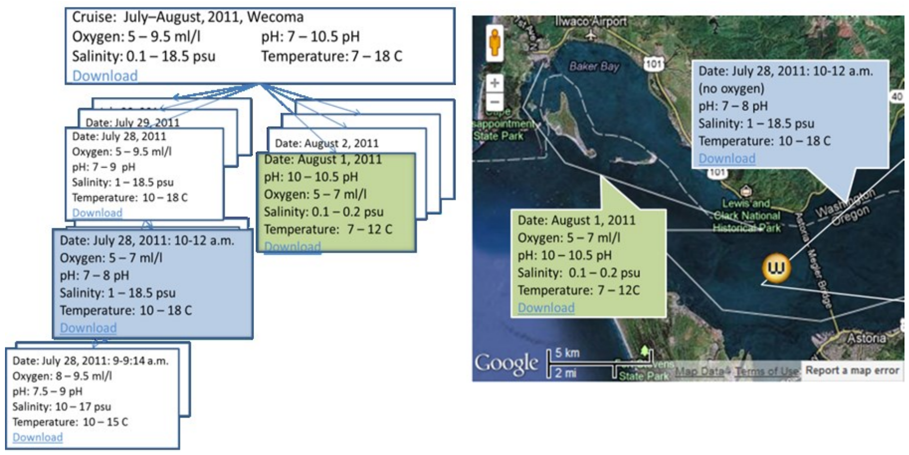


Fig. 3. Example of a dataset hierarchy: a dataset containing several million environmental observations (taken at 3 millisecond intervals) during a specific 2-month science cruise, segmented into a hierarchy. The white line on the map shows the cruise track, and the marker “w” shows the location of Lynda’s water sample. The most detailed level is a single simplified segment or leg of a cruise, often covering part of an hour or a few hours; these segments are aggregated into successively longer and more complex cruise segments, and lastly into an entire cruise dataset. The most relevant portion to Lynda is shown shaded on the left in the hierarchy, while the most relevant portion to Joel is shown shaded on the right. From [13].

3 Performance Study

3.1 Filter-Restart

While most searches of the current CMOP archive take a few seconds, we are interested in understanding the effects of further growth on interactive response times. While replicating the catalog can handle larger numbers of users, the number of catalog entries could increase by a factor of 100 in the coming years, which could increase response times for searches. In our prototype, the search engine's tasks of performing the search and identifying the top-k results account for the majority of the elapsed time seen by the user, and thus are our focus. This section briefly presents some of techniques we have explored to improve search performance, based on pre-filtering and exploiting the dataset hierarchy. (We have also investigated what affect the choice of hierarchy has on search efficiency, but do not report those results here [13].)

We extend the basic algorithm we use to identify the top-k results for a search to add a cutoff filter. We further refine that method with relaxation – a technique that reduces the filter's cutoff score if too few results are returned and re-issues the search. This combination of a filter and relaxation is called filter-restart.

3.2 Background

Our goal is to maintain search response in an "interactive response" range, while we grow the size or complexity of the underlying metadata collections. One area we investigated is query evaluation techniques to improve top-k query performance [8]. We wondered if these techniques could be applied to our metadata catalogs containing hierarchical metadata.

Metadata Catalog versus Index. Classic Information Retrieval systems primarily achieve good performance by using inverted indexes, where each word in a document is treated as a key, and the documents in which the word occurs are stored as values. In contrast, our datasets may contain real numbers and searches are generally for ranges rather that for a specific value, so an inverted index is not an appropriate indexing technology. Our catalog is index-like; however, the hierarchical structure is not driven primarily by performance considerations; instead, the hierarchy structure carries semantic meaning. However, these semantics can help search. For example, we have defined a parent dataset as containing the bounds of all its children. If the closest edge of a parent dataset is "far" from our search, we know that no child dataset can be close; if a parent is completely inside our search area, all children will be, too. Where practical and effective, we would like to use indexing concepts with our metadata to improve performance.

We implemented our catalog in PostgreSQL with B-tree indexes for non-spatial data and R-trees from PostGIS. Our search queries use the database indexes to navigate our catalog entries. We expect the majority of catalog accesses to come from identifying the top-k similar entries to a search; but our similarity function does not easily translate into a query against an index. Rather than a request for a specific set or range of identified rows, we need the top-k entries subject to a function that considers the center and the end-points of the data range, relative to another range given query time. Thus, while index algorithms inform our work, search performance over our catalog is not directly predictable from index performance statistics. There are similarities between our approach and, for example, Hellerstein and Pfeffer's RD-trees ("Russian Doll" trees) [7]. An implementation of our system using RD-tree indexes might provide faster performance than the current combination of B-trees trees and R-trees, though perhaps not by the orders of magnitude we desire.

Several other scientific fields have developed specialty indexes for scientific data, with the most advanced work in the field of astronomy [10–12, 16, 17]. Unlike our case, these systems generally assume they are dealing with homogeneous data.

Top-k Evaluation Techniques. Our search engine latency is primarily driven the execution time of (possibly multiple) SQL queries to identify the top-k results. We could process the entire metadata collection, computing the score of each entry, and then sort the resulting list. This approach does not scale well, and the sort is a blocking operation. These issues have led to development of improved top-k techniques and algorithms. We wished to explore the use of such top-k techniques and adapt them to our setting. We examined the classification of top-k approaches by Ilyas et al. [8] and used their taxonomy dimensions to identify those that apply to our setting. In particular, for the query-model dimension, we use a top-k selection query; we have data and query certainty; our infrastructure provides both sorted and random data access methods; we desire an application-level technique; and we use only monotone ranking functions. However, most the techniques identified there had limitations relative to our requirements [13]. The most appropriate seemed to be Filter-Restart techniques such as that of Bruno et al. [3], which operate by applying range-selection queries to limit the number of entries retrieved. A selection query is formulated that only returns entries above an estimated cutoff threshold (the *filter*), and the retrieved entries are ranked. Note that if the threshold is underestimated, too many entries are retrieved and performance suffers. Conversely, it may be overestimated, in which case too few entries are retrieved, and the query must be re-formulated and re-issued with a lower threshold (the *restart*). Filter-restart techniques rely for their performance improvement on the filter reducing the number of entries returned without compromising the quality of the result.

Filter-Restart and Cutoff Scores. It has long been known that limiting the results returned from a query by using a filter can dramatically improve response times [4]. To reduce the number of rows returned from the database, we add a filter to the search to return a subset of the rows guaranteed to contain the top-k results, by applying a formula derived from our scoring formula and adding a cutoff threshold. To be useful, the filter must be selective, and the time to evaluate the filter and process the results should be less than just evaluating all the rows without the filter.

A challenge for the filter-restart class of techniques is determining how to initially set the cutoff threshold for a given top-k search. As Chaudhuri et al. noted [5], there is as yet no satisfactory answer to estimating result cardinalities and value distributions in order to identify an optimal cutoff value, leaving this estimation as an open area for research. Bruno et al. translate a top-k query into a range query for use with a traditional relational RDBMS [3]; they perform this translation for three monotonic distance functions, SUM, EUCLIDEAN DISTANCE and MAX. They use database catalog statistics to estimate the initial starting score, using certain distribution functions.

Their experiments show that assuming uniformity and independence of distributions was computationally expensive and leads to many restarts [3]. They also note that "no strategy is consistently the best across data distributions. Moreover, even over the same data sets, which strategy works best for a query *q* sometimes depends on the specifics of *q*." For our primary use case (observational data), it is more common for attributes to be positively or negatively correlated than for them to be independent. Our data may also be strongly clustered, temporally, spatially, or on some observational variable. Thus, we expect to see issues similar to those found by other researchers in identifying the best cutoff value to use.

When an initial cutoff score fails to return sufficient matches (defined by a range such as $10 \leq k \leq 50$) , the search must be restarted with a revised cutoff score. This technique is often called *relaxation*, which is defined by Gaasterland as "generalizing a query to capture neighboring information" [6]. As with the initial cutoff score, we must identify a revised cutoff score to use for the revised search.

3.3 Basic Algorithm with Filter

Our basic algorithm first processes the given search terms to build an SQL query to retrieve entry information from the metadata tables. We then retrieve all roots from the database and score them. Any entry we deem likely to have "interesting" children we add to a list of parents; we then produce SQL to retrieve the children of parents in the list, and repeat the process at each next level of the hierarchy. When we get no rows returned from the database or we find no entries with likely interesting children at some level of the hierarchy, we terminate. Thus, rather than a single top-k query we issue an SQL query for (potentially) each level of any hierarchy within our forest of hierarchies. The scored entries are sorted and the top-k returned.

At first glance it may appear that we will score every entry in the catalog. If all entries in the catalog are roots (and therefore have no children), then a table scan is required. In contrast, if the ranges of the relevant variables in a parent entry are completely within the search term's range, we deem the children not interesting, and only return the parent. If the closest and furthest edges of the variable's range have the same score, we also deem the children not interesting, as the children's scores cannot be higher than the score of the closest edge of the variable. Again, only the parent entry is returned. In these cases, only a subset of the entire catalog's entries is scored.

The description so far is without the filter. The filter must operate over interval summaries and avoid false negatives. Therefore, for a particular cutoff score (*lowestScore*), our filter must return (at least) all entries that would achieve a higher score using our similarity measure. If insufficient matches are found from an application of the filter, we must restart the query with a "looser" filter. To implement the filter, we add terms to the SQL query to limit entries returned to:

- Those we estimate will have a final score for the entry that is higher than the cutoff score; that is, entries where the center of the data range has a score of *lowestScore* or higher. We may increase a summary's score if the data range overlaps the search term; however, we never decrease a score below that of the center of the data range. This filter term excludes entries that will not be adjusted upwards and that cannot have a score higher than *lowestScore*.

- Those whose children may have higher scores than the cutoff score; that is, entries that have children and where the closest edge to the search term has a score of *lowestScore* or higher. These parents may have a child with scores higher than the parent (if, for example, a child's bounds are near the parent's closest edge). If the closest edge has a score lower than *lowestScore*, not only will the parent not have a possible score higher than *lowestScore*, but none of its children will either; thus, "far" parents and their children are not retrieved.
- Entries that span at least one search term; or, where the closest edge has a score higher than *lowestScore*. These entries will have their scores adjusted upwards based on the amount of overlap with the search term. They also may have children whose bounds are completely within the search term, and thus are perfect matches for that term.

For geospatial search terms, we developed a slightly different filter, in order to reduce the number of heavyweight spatial calculations. In addition to storing the entry's shape, we add two columns to our data model. While building the metadata entry, one column is updated with the shape's centroid, and the other with the maximum radius of the shape, allowing us to quickly bound the minimum distance to the shape.

Since performance (in terms of elapsed time and resources used) is driven largely by the number of rows returned from the SQL query, applying the filter should improve performance. The first SQL query in any search only evaluates the root entries in the forest of trees. We access successive levels of the hierarchy (i.e., children) by parent_id, and the filter acts to reduce the number of child entries returned from each level. Our initial cutoff score is a default value that empirically works will for our archive. However, if the cutoff score is set too high to find *k* entries with equal or higher scores, insufficient (or no) rows will be returned from the initial walk through the hierarchy, and a restart is required.

3.4 Relaxation

If fewer than *k* entries are found with sufficiently high scores, we restart the search with a lower starting score, that is, we *relax* our filter. We developed and experimented with three relaxation techniques:

- Naïve: In our simplest technique, whenever we do not find sufficient results at our starting filter score, we reissue the query after having reduced the filter score by a fixed (arbitrary) number.
- Adaptive: Our second technique uses information calculated while processing the entries returned from a query to adjust (in our case, lower) the starting score for subsequent queries.
- Contraction: This technique modifies adaptive relaxation by reversing relaxation. That is, if we discover that we are retrieving more than the desired number of results using a specific filter score, we increase the score for subsequently issued SQL queries during the same search.

3.5 Performance Tests

Methods and Data. We tested filtering and the three types of relaxation: Naïve (N), Adaptive (AD), and Contraction (CN). We focus on the performance of a single server: if single-server performance is sufficient for the expected workload, no additional scaling approaches are needed, while if more servers are needed, the number will be defined largely by the single-server performance. To understand the performance characteristics of our approach apart from the (potentially confounding) impact of underlying disk hardware, we restrict ourselves to data sizes where the working set of the metadata catalog can fit in memory (2GB of buffer pools). We created two collections over which to run our tests. Both summarize real-world data from a non-CMOP archive of interest to our scientists. The smaller is approximately 5 times the size of CMOP's current collection, while the larger is 25 times the size. (We have also experimented with a dataset 200 times larger.) We implemented the search evaluation techniques described earlier, while retaining the current design of Data Near Here.

To test our techniques over different hierarchy "shapes", we developed 8 hierarchies, each with different aggregation characteristics. In addition, we keep a ninth "no hierarchy", with every leaf treated as a root. We developed three test suites of searches, using search terms that we know from experience to be frequently used by our scientist and with very different performance characteristics: time-only, space-only, and time-and-space. We report here some of our results with the time-only and space-only suites. Each query in a test suite was run 5 times against each target hierarchy. For each search, the maximum and minimum response times were discarded and the response times for the remaining three queries were averaged, as is common in database performance measurement. Whenever a different table or hierarchy was accessed, the first search was discarded, to force the relevant indexes (and, possibly, data) to be loaded into memory, simulating a "warm cache".

All tests were run on a 2 quad-core 2.13 GHz Intel Xeon system with 64 GB main memory, running Ubuntu 3.2.0 with Apache 2.2.22, PHP 5, PostgreSQL 9.1 and PostGIS 2.0. The PHP space limit per request was increased to 2 Gb, and the time limit to 1,000 seconds. PostgreSQL 9.1 only uses a single core per user request and the flow through the application is sequential; as a result, a search uses only one processor.

Test Collection. We scanned two sets of data from NOAA for use in the performance tests.[2] The first two are chlorophyll-a concentration data from NASA's Aqua Spacecraft[3], in two different formats: Science Quality (herein abbreviated to Collection *s*) and Experimental (herein referred to as Collection *e*). The data is gathered by the Moderate Resolution Imaging Spectroradiometer (MODIS). The data scanned differs in density (0.05° for Science Quality versus 0.0125° for Experimental), and has differences in the algorithms used to produce the data.

We created a configurable scanner that reads the subset of data for our geographic area of interest (the region between latitude 40 to 50, longitude -122 to -127), and creates a leaf dataset record for each (configurable) block. The blocks were configured to be 0.25° latitude by 0.25° longitude; each block is treated as a separate "da-

[2] To reduce confusion, we will refer to the results of scanning each of these sets of data as a "catalog".

[3] http://coastwatch.pfeg.noaa.gov/coastwatch/CWBrowser.jsp

taset" from the perspective of the metadata collection. Where there are missing values for cloud cover and other reasons, the actual area of each block represented in the data may be substantially smaller than the nominal area. The scanner checks the physical locations of the data points reported, and represents the physical extent of the valid data by a convex hull of the valid data points. The leaf data counts for Collection *s* are 192,554 leaf records, with 962,770 entries in the variables table (5.5 times the size of our current catalog). Collection *e* has 930,261 leaf records, with 4,653,105 entries in the variables table, or 4.8 times the size of *s*.

Test Hierarchies. We developed 8 different hierarchies, and a configurable hierarchy creation program. The hierarchies represent different patterns of aggregation of space and time, with the number of levels varying from one (all entries are leaves) to 5 (root, three intermediate levels, and leaf). Each hierarchy was built over the *s* and the *e* collections. In the performance tables below, a designation "5e" means the *e* data with Hierarchy 5; similarly, "5s" means the *s* data with Hierarchy 5.

Search Suites. Since almost all searches include time or space or both search terms, we developed three search suites: one with queries containing only a time search term; one with queries with a space search term only; and one with queries containing both a time term and a space term. We present some results for the first two here.

Time search suite. This search suite consists of 77 searches. The searches range in duration from 1 day to a search spanning 44 years (1/1/1970 to 1/1/2014), which is longer than our entire temporal coverage. Some searches were defined that match exactly the various levels of the hierarchies; others are offset from and overlapping hierarchy groupings, but covering the same durations; and some are both longer and shorter than various hierarchy grouping levels.

Space search suite. This suite consists of 20 searches. The smallest search's spatial extent is 4.4 sq.Km, while the largest search spans 2 degrees latitude by 2 degrees longitude for an area of approximately 52,000 sq.Km., which is a bit more than 8% of our entire geographic coverage area. As with the time searches, spatial searches were created that aligned with hierarchy groupings, were offset from or overlapping hierarchy groupings, and were both larger and smaller than hierarchy grouping levels.

In all cases, we requested $10 \leq k \leq 50$, that is, a minimum number of 10 and a maximum of 50 entries to be returned from a search.

Time Search Suite Results. We first ran tests for the time suite to assess the benefit of the filter alone (without restart) [13]. Adding a filter without restart capabilities means that a search may not result in the minimum requested entries. For most of the hierarchies, adding the filter reduces the response time to one-fifth or less of the response time without the filter. For "None" and no filter, every search failed to complete (either exhausting PHP memory or time limits) for the *e* collection; when the filter was added, nearly four-fifths of the searches completed, although the mean response time for those queries was very high compared to the other hierarchies. For the other hierarchies, almost all searches for the *e* collection completed.

Space Search Suite Results. For consistency, we tested the space search suite against the same collections and hierarchies as used for the time search suite. Many of the "no filter" searches did not complete, so we do not include those results. We report results for the *e* collections while including one *s* collection (2s) for comparison. Of the 20 searches in the space suite, 7 resulted in more than one iteration using naïve relaxation; four searches had two iterations only. Three searches had more than 2 iterations;

one search iterated 8 times before exiting and returning fewer than the minimum desired results.

For the searches with two iterations (that is, the searches that might be affected by AD), AD used the same final cutoff score as Naïve relaxation; this means that the revised cutoff calculated by AD after the first iteration was either higher than the "minimum cutoff adjustment", or the same as Naïve's cutoff score.

Table 1 summarizes the total number of iterations, average rows returned from SQL, the average number of matches returned from the searches, and response times for the three searches with more than two iterations. All response times are significantly longer than for the time-search suite, as the spatial comparisons used in the distance measure take more time to calculate than the numeric comparisons used for one-dimensional variables such as time. In every case, AD reduced the number of iterations and reduced the response time by more than 50%, even when the number of rows returned from the database (Avg. SQL Rows) was almost identical to those returned for N. This is because the *lowestScore* was adjusted down by more than naïve_reduction for all affected searches, and in most cases the search found sufficient matches during the first restart.

Table 1. Space Search Suite, N vs. AD: Summary of (Three) Searches with More Than Two Iterations. "# Iterations" is the total number of iterations across all searches in the suite.

Hierarchy	Naïve (N)				Adaptive (AD)			
	# Iterations	Avg. SQL Rows (1000's)	Avg. Matches Found	Mean Response (s)	# of Iterations	Avg. SQL Rows	Avg. Matches Found	Mean Response (s)
1e	17	20,419	36	33.8±6.2	7	20,229	24	14.4±2.2
2s	19	7,688	33	14.2±3.5	6	7,444	15	4.5±0.2
2e	17	33,302	36	61.6±7.4	8	33,074	31	29.9±2.4
3e	17	22,505	35	31.5±5.3	7	21,754	24	13.2±2.9
5e	17	22,484	35	32.9±4.3	7	21,732	24	13.8±3.2
6e	17	22,984	35	32.9±4.3	7	22,232	24	13.8±3.2

Table 2 reports the average rows retrieved by SQL and response time for the subset of searches affected by CN for the space-search suite, when run against different hierarchies. As shown in Table 2, CN increased the final cutoff scores for 13 of the 20 space queries for hierarchy 1e, but only for between 4 and 7 searches over the other hierarchies. We see that CN results in lower mean response times and lower variability (lower standard deviations) for affected searches in every hierarchy tested. CN reduces the number of rows processed by approximately 20% to nearly 50%, while reducing response time by approximately 50% and reducing variability in all cases.

We conclude that for this search suite, the combination of the filter with adaptive relaxation and contraction provides the best results across all hierarchies (as it did for the other search suites [13]). While these techniques do not help all searches, they also do not negatively impact the searches that they do not improve. As multiple iterations tend to be associated with long response times, the searches with the longest response times tend to be the ones most improved.

Discussion. The overall goal in our performance work is to understand, improve and predict the performance characteristics of our approach. We showed that as the metadata collection size increases, our filter allows us to complete searches that would otherwise time-out or exhaust memory. We have experimented with an additional, modified filter for space searches that may provide additional benefits by allowing use of spatial indexes (which are not used by our current spatial measure); however, the improvement achieved of approximately 1/3 is less than the order-of-magnitude needed to make a substantial difference to the scalability.

Over all search suites (including time-space), less than one third of the searches required restart. For the time search suite, restarts only occurred in one hierarchy. For searches that required restart, Adaptive (which estimates a revised cutoff score based on the matches found so far) reduced the number of restarts required by half or more over Naïve (which uses a fixed revision to the cutoff score). For affected searches, AD reduced response time by around half, and reduced the variability of response times across the searches within the search suite by around half as well. Thus, Adaptive relaxation achieves its goal of reducing latency over Naïve. For all suites, adding Contraction (successively increasing the cutoff threshold as the number of desired matches is reached) reduced the response time and variability; the improvement was around 20-25% for the affected searches. However, the improvement gained by adding Contraction was lower than the improvement of Adaptive over Naïve.

Based on these and other tests, we see that hierarchies can, depending on their organization and on the specific search, further improve performance compared to no hierarchy. For most hierarchies compared, the increase in response times between the *s* and *e* collections was, for most searches, smaller than the increase in the number of entries between the collection sizes. This reduction supports the added value of taking advantage of hierarchies to improve response times for a given collection size.

In the currently deployed version of the search engine, we use the filter with Adaptive relaxation and Contraction, and the centroid-and-max-radius based spatial filter.

Table 2. Space Search Suite, AD vs. CN, for searches affected by CN only. "# Affected" is the number of searches with increased low score.

Hierarchy	# Affected	Adaptive (AD)			Contraction (CN)		
		Avg. SQL Rows (1000's)	Avg. Matches Found (1000's)	Mean Response (s)	Avg. SQL Rows (1000's)	Avg. Matches Found (1000's)	Mean Response (s)
1e	13	115.2	39.2	35.0±58.4	82.1	3.5	29.6±24.8
2s	5	65.4	33.3	21.3±6.3	44.1	9.0	14.7±3.6
2e	7	239.5	114.5	78.2±69.3	159.0	30.4	54.3±34.3
3e	5	286.7	147.6	93.9±44.2	212.8	91.2	66.4±25.4
5e	4	333.3	181.8	110.6±48.4	272.1	128.7	87.0±23.8
6e	4	333.7	181.8	110.6±48.4	272.5	128.7	87.0±23.8

All search types perform well for our current catalog size, and for the expected organic growth of our catalog over the next 5 years. To support larger archives, we may need to modify how we store metadata on variable ranges, and learn to distinguish and replace hierarchies that lead to long response times.

4 Further Challenges

We cover here additional issues in dataset searching.

- Selecting a good initial cutoff score can avoid restarts, and it might be possible to use metadata catalog statistics to guide the choice. Bruno et al. [3] attempt to adapt to the underlying data by taking a sample of the data, and running a training workload on it to identify a cutoff score. Other queries from a similar workload can then use the same cutoff score. To use this technique, we would need a "search similarity function" that can compare the current search to other searches, then use the cutoff score from the most similar search. There might be other techniques for selecting a good initial cutoff.
- Further refining the filters we construct could cut down on the number of dataset summaries for which we have to compute summaries. Currently, the filter is constructed of "per term" conditions that operate independently. There might be advantages to constructing filters that consider multiple terms jointly.
- An important question, given the widely different performance results from the search suites, is how to predict which hierarchies will provide the good performance for queries or query classes, if there is more than one choice. We explored a range of hypotheses for how to predict the best hierarchy to use, but without clear success [13]. The best-performing hierarchy varies according to the type of search terms and the density of data close to the combination of terms. If we cannot accurately determine a good hierarchy a priori, there are options to consider:

  (a) Start a search on each hierarchy and provide the response from the hierarchy that comes back first, canceling the others.
  (b) Choose a single "good enough" hierarchy approach as a default, and further develop performance methods to maintain performance in acceptable ranges.
  (c) Keep every set of catalog tables small enough to give fast response from each, and add an integrator to combine the top-k from across the various lists.
  (d) Provide partial results back early once a search with long response times is recognized (e.g., many results are returned from an SQL query, or a restart is required), and continue to refine them as more results become available.
  (e) Try additional methods to predict hierarchy performance.

- As noted, we wish to understand the performance of a single server; if single-server performance is sufficient for the expected workload, no additional scaling approaches are needed. When single-server performance is found insufficient even after optimization, throughput can be further scaled by adding servers. In our current archive, we use different hierarchies for different types of data, mixing all of them in a single metadata collection. Thus, a single search may be traversing many different hierarchy shapes simultaneously, which could be parallelized. It is a bit harder to partition a hierarchy for parallel search, as only a few branches of a hierarchy might apply to a specific search.

- An implicit assumption in our current similarity function is that data values are uniformly distributed in the regions and intervals captured in the summaries. For some features (such as time), that assumption is fairly accurate, but not for all. Initial tests show that our summary-based similarity function nevertheless closely tracks one that considers individual observations. However, incorporating more information about values distributions in a dataset, including joint distributions of correlated variables, might yield benefits.
- Another major performance aspect is the effect of varying the complexity of the queries themselves. We do not know the "usual" number of search terms, but that number will have a large effect on response times. User studies in Internet search have found 5 to be a common number for text retrieval [1, 2, 9]; but that number might not be typical in our setting. Further research might characterize dataset-search complexity, and develop techniques for accelerating complex queries.
- We know that retrieving data from the database is a large component of our search engine latency. Our initial data design was chosen for ease of implementation, but constrains our performance. We have a fairly general schema for variables that makes it easy to add a new variable, but requires joins when we have search terms for more than one variable. A materialized view that pivots the variables table save the join cost.
- One aspect of scaling we have not treated here is the diversity of variable naming and meaning as an archive grows. Especially as we try to support search over multiple archives, we will encounter different variable names that represent the same quantity, and the same name used for different quantities. We explore this problem and possible solutions in other work [14].

These issues mainly consider issues that arise in our IR-style approach to dataset search (though any approach will need to deal with the variable diversity issues mentioned in the last bullet). Obviously, there could well be other effective approaches. We are also interested in how our techniques can be adapted to other domains where temporal and geospatial aspects are not as dominant.

Given the growth in size and variety of scientific archives and the amount of time spent by scientists in locating relevant data, maintaining interactive response times will be a challenge. Our scientists have found significant value in the search capability we've already provided in Data Near Here. Scaling this initial work into a multi-repository search engine through IR-style data discovery can help increase the touchability and reuse of data that was previously "one touch". We thereby create higher returns on the investment in data collection and enable new scientific discoveries.

Acknowledgments. This work is supported by NSF award OCE-0424602. We thank the staff of CMOP for their support.

5 References

1. Ageev, M. et al.: Find it if you can: A game for modeling different types of web search success using interaction data. Proceedings of SIGIR. (2011).
2. Aula, A. et al.: How does search behavior change as search becomes more difficult? Proc. of the 28th international conference on Human factors in computing systems. pp. 35–44 (2010).
3. Bruno, N. et al.: Top-k selection queries over relational databases: Mapping strategies and performance evaluation. ACM Trans. Database Syst. TODS. 27, 2, 153–187 (2002).
4. Carey, M.J., Kossmann, D.: On saying "enough already!" in SQL. ACM SIGMOD Rec. 26, 2, 219–230 (1997).
5. Chaudhuri, S. et al.: Integrating DB and IR technologies: What is the sound of one hand clapping. CIDR'05. 1–12 (2005).
6. Gaasterland, T.: Cooperative answering through controlled query relaxation. IEEE Expert. 12, 5, 48–59 (1997).
7. Hellerstein, J.M., Pfeffer, A.: The RD-tree: An index structure for sets. University of Wisconsin-Madison (1994).
8. Ilyas, I.F. et al.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. CSUR. 40, 4, 11 (2008).
9. Jansen, B.J. et al.: Real life, real users, and real needs: A study and analysis of user queries on the web. Inf. Process. Manag. 36, 2, 207–227 (2000).
10. Koposov, S., Bartunov, O.: Q3C, Quad Tree Cube: The new sky-indexing concept for huge astronomical catalogues and its realization for main astronomical queries (cone search and Xmatch) in open source database PostgreSQL. Astronomical Data Analysis Software and Systems XV. pp. 735–738 (2006).
11. Kunszt, P. et al.: The indexing of the SDSS science archive. Astron. Data Anal. Softw. Syst. Vol. 216, (2000).
12. Lemson, G. et al.: Implementing a general spatial indexing library for relational databases of large numerical simulations. Scientific and Statistical Database Management. pp. 509–526 (2011).
13. Megler, V.M.: Ranked Similarity Search of Scientific Datasets: An Information Retrieval Approach (PhD Dissertation in preparation), (2014).
14. Megler, V.M.: Taming the metadata mess. IEEE 29th International Conference on Data Engineering Workshops (ICDEW). pp. 286–289, IEEE Computer Society, Brisbane, Australia (2013).
15. Megler, V.M., Maier, D.: Finding haystacks with needles: Ranked search for data using geospatial and temporal characteristics. In: Bayard Cushing, J. et al. (eds.) Scientific and Statistical Database Management. pp. 55–72 Springer, Heidelberg (2011).
16. Singh, G. et al.: A metadata catalog service for data intensive applications. Proceedings of the 2003 ACM/IEEE Conference on Supercomputing. p. 33 (2003).
17. Wang, X. et al.: Liferaft: Data-driven, batch processing for the exploration of scientific databases. CIDR. (2009).