

Technical Perspective

In-Situ Database Management

By David Maier

IMAGINE YOU HAVE a collection of data files—say, cell-tower call records—and you believe some of them might contain useful information: towers that are near capacity, numbers whose calls are frequently dropped, failed hand-offs. You want to run a few queries over selected files to explore which ones might merit further analysis and determine what kinds of knowledge you might extract. If a file is small, you might transfer it to your own computer and inspect it with a spreadsheet program or an analysis environment such as R. However, suppose individual files are too large to fit on your computer. What are your choices then for exploring them?

One possibility is to load the data into a database management system (DBMS) and use the query language—likely SQL—to ask your questions. While you get the advantage of a high-level data language, it might take hours to load the data before you can pose your first query. If you want to switch to another file, then you have to wait again while that file loads. Moreover, any file you do load now takes up at least twice the storage space, since its data is in both the database and on the file system. Deleting the original file might not be possible, if it has other users.

An option is to use a MapReduce framework, such as Hadoop, to run your preliminary analyses. Now your “time to insight” is delayed by having to write (and probably debug) a program. Even if you are able to formulate your questions as programs fairly quickly (perhaps using a language layer such as Hive or PigLatin), each query you run will scan the whole file anew. In addition, you lose performance enhancements such as indexes and optimization available in a DBMS.


Such unattractive trade-offs face nearly everyone wanting to quickly explore a new data source. The following paper by Alagiannis et al. investigates a third approach, extending a DBMS so

The following paper is exciting, as it minimizes upfront costs when exploring new data sources, and it opens up a wide range of additional techniques to pursue for in-situ data management.

it can use the file data in situ, without having to load it first. They term their approach “NoDB” to indicate it does not require a separate copy of the data stored internally to the DBMS.

Note that some DBMSs do support links to external files that are viewed as tables, which are parsed and temporarily loaded on demand. That approach does avoid the initial load into persistent storage and allows the loading process to overlap with other query stages. However, loading happens on every query, as with the MapReduce approach. Furthermore, such external data is a second-class citizen, lacking the indexes and statistics that speed performance on internal data. The NoDB approach, in contrast, tries to make in-situ data first class, by using an incremental, pay-as-you-go approach to providing DBMS functionality that tries to minimize up-front load costs, while capturing the work that is done for the benefit of later queries.

The authors built a specific instance of a NoDB system called PostgresRaw. The main techniques it uses are to avoid parsing portions of a file not needed by the current query, and reusing the work it does do via a “positional map” (a sort of structural index) that remembers the location of fields in records it does access. Thus, initial queries avoid the full load cost of an external file, while later queries take advantage of previous parsing work. The authors also consider incremental methods on in-situ data, such as collecting statistics and caching data from one query to another. The evaluation of PostgresRaw shows the performance advantage of NoDB technology over the alternatives mentioned here.

Should we expect our DBMSs in the future will do away with internal storage and run entirely over in-situ data? Likely not—the NoDB approach targets large, static datasets (though it could be extended to handle certain classes of updates, such as appending records). Data subject to small, frequent changes is best maintained by the DBMS. Also, if you determine at some point that a file will be intensely queried in the future, then it is worthwhile to incur the up-front load cost, in exchange for faster queries. Nevertheless, the paper is exciting, as it minimizes up-front costs when exploring new data sources, and it opens up a wide range of additional techniques to pursue for in-situ data management, such as incremental value-based indexing, synthesizing access methods for common file types, and selective transfer of in-situ data to internal storage. 

David Maier (maier@cs.pdx.edu) is Maseeh Professor of Emerging Technologies in the Department of Computer Science at Portland State University, Portland, OR.

Copyright of Communications of the ACM is the property of Association for Computing Machinery and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.