

Data Management

"Data Management" design considerations emerge when we consider the data management aspects of a complete workflow system. The amount of information available to us is literally exploding, and the value of data as an organizational asset is widely recognised. Otherwise, data can become a liability, with the cost of acquiring it and managing it far exceeding the value derived from it.

To get the most out of the large and complex data, users require tools that simplify the tasks of managing the data and extracting useful information. Since the objective of scientific experiments is to transform data in a meaningful way, to the user. A common approach is to model the input and user requirements of user components (schemas) as tables in a database system.

A database management system or DBMS is software designed to assist in maintaining and utilizing large collection of data. The need for such systems, as well as their use, is growing rapidly. The alternative to using a DBMS is to store the data in files.

Furthermore, most of the data available on the web or other sources is readily stored in relational databases; complete and accurate data models can hence be designed to accommodate many diverse information sources. The management system should support any combination of tuple attributes that is supported by the underlying database management system. In this case, any pairing of any number of SQL data types. Storing scientific data as tuples in a relational database also opens the discussion to partially processed (or partially generated) experiment points. The issue of partial data can be furthermore split into two categories. row-wise and column-wise.

Row-wise partial data is tracked at the experiment scope. When we observe that scientific experiments are usually run on sets of inputs composed from database tuples, rowwise partial results refer to the ability of the management system to track the experiment's execution, and provide information about it in terms of the data computed thus far. In order to fully support this feature, the system must not be limited to returning merely %completion information, and must support querying the computed tuples out of the total set. This feature is most important for experiments such as search-based optimizations, and iterative improvement algorithms.

Column-wise partial results refer to partially computing the tuples' constituent information, while still considering the entire experiment point set. This feature is clearly different from row-wise partial results, as it omits attributes of the data points, but not the data points themselves. This form of partial results becomes very useful when considering certain cases of information retrieval applications. A user component designed to access a bioinformatics publication database such as PubMed might retrieve a large feature set such as the title, abstract, and body of the requested article. However if, for the purpose of the experiment, the user only requires the first two features, retrieving the body would be a waste of resources as it is typically much larger than the first two attributes combined.

The management system should be configurable to allow such interpretations, to the extent of the component's support of the feature. Another feature essential to data management is tracking the data through

the chain of execution, especially in complex experiments. Having access to the complete chain of data transformations including branches, joins, and choice of conditional paths does not only ease debugging, but is also vital to interpreting experiment results. To be of utmost use, tracking must operate at the granularity of individual tuples in the experiment point set, and allow distinction between outputs even though there might be temporary mismatches between the number of input points and the number of output points.

Data tracking should not be confused with experiment point caching and storing. Many workflow management systems implement experiment point caching and claim to have both features. Tracking experiment points introduces the concept of 'varying multiplicity' data. This term describes cases when one experiment point computes a result that actually corresponds to more than one experiment point in the range.

An example of such an experiment is a 'scatter-gather' operation, popular in information retrieval. Here, the computations are staged, so that the first stage computes a multiset from a given experiment point, which could then be arbitrarily separated and re-grouped in the next stage (hence the need for modeling the range as multiple experiment points). A typical workflow solution is to explicitly program this feature but this approach still requires us to distinguish between gathering (collector) operations and regular one-to-one experiments, and is often relegated to the user's component.

As one of the main objectives of the workflow management system is to hide such complexity, this feature should also be mandatory. An oft-overlooked feature in scientific workflow management systems is eliminating the difference between already-computed data and information coming from experiments that are not yet run. In a system supporting this feature, the user could just 'query' for the experiment results regardless of whether they have been already computed. If the results are not available, the query effectively acts as a cue for requesting that the data be freshly computed or materialized. However, care must be taken when using such a feature to ensure that the target execution context supports such automatic computation in a feasible manner.

Consider, for example, the following query:

```
SELECT * FROM sentences WHERE sentence.length > 3;
```

This hypothetical experiment is meant to be performed using a component that generates sentences of a certain length. While typical query processing engines would trap 'unsafe' queries, or re-order plans to improve efficiency of execution, it is unclear if the above query will terminate in a reasonable amount of time or constitutes an abuse of experiment specification flexibility. The fundamental problem here is that query optimization functionality does not extend into the realm of user-supplied codes; this feature hence places significant onus on careful experiment design.

Closing the discussion on data management requirements, we arrive at the issue of execution priority at experiment point level. A flexible management system should support changing the order of execution in the set of experiment points defining the input, as long as there are no dependencies between the targeted experiment points. For instance, we should be able to employ algorithms that automatically infer priorities based on knowledge about the domain and individual experiment point requirements.