Dataflow Programming for Big Engineering Data – extended abstract –

Felix Beier, Kai-Uwe Sattler, Christoph Dinh, Daniel Baumgarten
Technische Universität Ilmenau, Germany
{first.last}@tu-ilmenau.de

Nowadays, advanced sensing technologies are used in many scientific and engineering disciplines, e. g., in medical or industrial applications, enabling the usage of data-driven techniques to derive models. Measures are collected, filtered, aggregated, and processed in a complex analytic pipeline, joining them with static models to perform high-level tasks like machine learning. Final results are usually visualized for gaining insights directly from the data which in turn can be used to adapt the processes and their analyses iteratively to refine knowledge further. This task is supported by tools like R or MATLAB, allowing to quickly develop analytic pipelines. However, they offer limited capabilities of processing very large data sets that require data management and processing in distributed environments – tasks that have vastly been analyzed in the context of database and data stream management systems. Albeit the latter provide very good abstraction layers for data storage, processing, and underlying hardware, they require a complex setup, provide only limited extensibility, and hence are hardly used in scientific or engineering applications [ABB+12]. As consequence, many tools are developed, comprising optimized algorithms for specialized tasks, but burdening developers with the implementation of low-level data management code, usually in a language that is not common in their community.

In this context, we analyzed the source localization problem for EEG/MEG signals (which can be used, e. g., to develop therapies for stroke patients) in order to develop an approach for bridging this gap between engineering applications and large-scale data management systems. The source localization problem is challenging, since the problem is ill-posed and signal-to-noise ratio (SNR) is very low. Another challenging problem is the computational complexity of inverse algorithms. While large data volumes (brain models and high sampling rates) need to be processed, low latencies constraints must be kept because interactions with the probands are necessary. The analytic processing chain is illustrated in Fig. 1. The Recursively Applied and Projected Multiple Signal Classification (RAP-MUSIC) algorithm is used for locating neural sources, i. e., activity inside a brain corresponding to a specific input. Therefore, 366 MEG/EEG sensors are placed above the head which are continuously sampled at rates of 600 – 1250 Hz. The forward solution of the boundary element model (BEM) of the brain at uniformly distributed locations on the white matter surface is passed as second input. It is constructed once from a magnetic resonance imaging (MRI) scan and, depending on the requested accuracy, comprises 10s of thousands of vertices representing different locations on the surface. RAP-MUSIC recursively identifies active neural regions with a complex pipeline for preprocessing signal measures and correlating them with the BEM. To meet the latency constraints, the RAP-

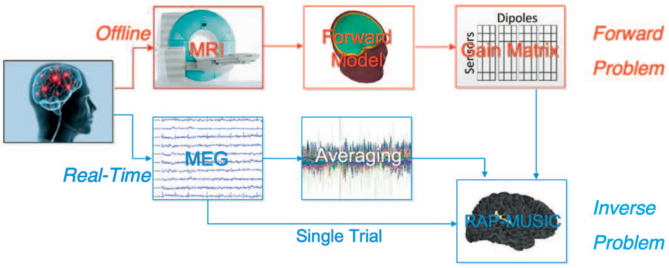

Figure 1: Overview Source Localization Processing Chain

MUSIC algorithm has been parallelized for GPUs and a C++ library has been created in an analysis tool called MNE-CPP [DLS+13], including parsers for data formats used by vendors of medical sensing-equipment, signal filter operators, transformation routines, etc. Although this library can be used to create larger analyses pipelines, implementing and evaluating new algorithms still requires a lot of low-level boilerplate code to be written, leading to significant development overheads. Latter can be avoided with the usage of domain-specific languages which are specialized for signal processing, natively working on vectors or matrices as first-class data types like MATLAB. But they offer less control over memory management and parallelization for custom algorithms which are crucial for meeting the latency constraints. When large-scale data sets shall be processed, even a specialized tool quickly runs into performance problems as distributed processing is mostly not supported because of large development overheads for cluster-scale algorithms.

To handle these problems, we propose to apply dataflow programming here. We implemented a multi-layered framework which allows to define analytic programs by an abstract flow of data, independent from its actual execution. This enables quick prototyping while letting the framework handle data management and parallelization issues. Similar to Pig for batch-oriented MapReduce jobs, a scripting language for stream-oriented processing, called *PipeFlow*, is provided as front-end. In the current version, *PipeFlow* allows to inject primitives for partitioning dataflows, executing sub-flows in parallel on cluster nodes leveraging multi-core CPUs, and merging partial results. We plan to automatically parallelize flows in the future using static code analysis and a rule-based framework for exploiting domain-specific knowledge about data processing operators. The dataflow programs are optimized by applying graph rewriting rules and code for an underlying execution backend is generated. Therefore, the framework provides an engine called *PipeFabric* which offers a large C++ library of operator implementations with focus on low-latency processing. One key aspect of *PipeFabric* is its extensibility for complex user-defined types and operations. Simple wrappers are sufficient to embed already existing domain-specific libraries. For our use case, processing of large matrices is required. Therefore, we used the Eigen library and are currently porting functions from MNE-CPP. We are also working on code generators for other back-ends like Spark which will be useful for comparing capabilities of different frameworks for common analytic workloads – which, to the best of our knowledge, has not been done yet.

References

[ABB+12] I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A. Ailamaki. NoDB: efficient query execution on raw data files. In *ACM SIGMOD*, 2012.

[DLS+13] C. Dinh, M. Luessi, L. Sun, J. Haueisen, and M. S Hamalainen. Mne-X: MEG/EEG Real-Time Acquisition, Real-Time Processing, and Real-Time Source Localization Framework. *Biomedical Engineering/Biomedizinische Technik*, 2013.