M491: Theoretical Basics of Big Data Analytics and Real Time Computation Algorithms

Course Goals

The main goal of this course is to provide students with a unique opportunity to acquire conceptual background and mathematical tools applicable to Big Data Analytics and Real Time Computation. The course will briefly review specific challenges of Big Data Analytics, such as problems of extracting, unifying, updating, and merging information and specific needs in processing data, which should be highly parallel and distributed. With these specific features in mind we will then study more closely a number of mathematical tools for Big Data analytics, such as regression analysis, linear estimation, calibration problems, real time processing of incoming (potentially infinite) data. We will see how these approaches can be transformed to conform to the Big Data demands. We will also discuss why most of widely used algorithmic languages are not quite appropriate for solving such problems and outline alternative approaches.

Course Ideas

Within traditional approaches to information processing we have to collect all the data in one array and apply a processing algorithm to it. If raw data is distributed among many sites and its total volume is large it immediately leads to certain technical problems:

- Accumulating all the raw data in one place would require excessive storage resources.
- Feeding huge arrays of data to an algorithm would require lots of operating memory and computing power.
- The idea of processing all the data at once does not reveal (and actually hides) possibilities for parallel or distributed computing.

In the course it is shown that instead of collecting together all the raw data and process it all at once we can naturally split the whole process into simple highly independent pieces. Specifically:

- Extract certain sufficient information from each instance of raw data and represent it in a convenient "canonical" form.
- Combine pieces of canonical information.
- Update accumulated canonical information when a new instance of raw data becomes available.
- Obtain a final result from accumulated information in canonical form.

It turns out that often information in canonical form has a fixed size, which does not depend on the amount of raw information used to produce it. As a result all the separate steps of extracting canonical information from raw data, combining it and obtaining a final result do not require excessive amounts of memory or computing power. After two pieces of canonical information are combined into one they can be immediately discarded. Extracting and combining pieces of canonical information can be performed on different computers without any need of synchronization. That provides a wide range of natural options for massive parallel distributed computing.

Course Topics

Real time learning.

Example: Linear estimation based on calibration.

Regression algorithm, which is used to process a stream of data in real time is permanently updated by itself by another stream of "calibration" data.

Problems:

- Amount of calibration data may be huge and rapidly growing. As a result collecting and storing all the calibration data may require lots of storage.
- Besides, computing an updated version of regression algorithm based on these data would require a lot (and even constantly growing) amount of time.

Solutions:

- When calibration measurement arrives – do not store it, but use it to update specific information (of a fixed size), which is sufficient for computing the regression algorithm. Such "packed" information is analogous to sufficient statistics.
- Do not re-compute the regression algorithm "from scratch" but update it using only the new calibration data and stored sufficient information.

In the course: Consider the linear calibration problem, study its computational complexity using "straightforward" and "minimal sufficient" approaches.

Real time signal processing.

Standard approaches would require to record a complete signal (for example sound recording) and then apply to it a processing algorithm. Although such approach can provide the most accurate processing, it cannot be done in real time since it would require to a record full signal (or big chunks of it).

However if time is critical, processing can be performed as the data arrives, using a "sliding window". Such approach would not require storing the source signal and would provide a feasible balance between processing quality, complexity and delay.

In the course: Consider a simple optimal processing problem for a (potentially) infinite array of data and develop an optimal algorithm for a given computational requirements and delay.

Decomposition of a big problem into a set of similar, but smaller problems, which can be solved in parallel. To design an appropriate algorithm that would benefit from such decomposition one would need to define explicitly how to compute the corresponding parts in parallel, or use an appropriate high level programming language (see next section).

In the course: Consider several examples, such as vector and matrix multiplication. Time requirements for "standard" and "parallel" algorithms.

Thinking Big.

Contemporary algorithmic thinking is severely linked to a programming language. At the same time almost all algorithmic languages, which are used for applied problems explicitly determine the order of operations. As a result, even if a computer can handle multiple problems at once a classical algorithmic language would not allow using parallelism implicitly. To overcome that a programmer should explicitly determine which parts of code can be executed in parallel.

A radical approach to that problem is move towards algorithmic languages that do not specify the order of operations (such as, e.g. functional languages ML, Haskell, etc.).

In the course: Short introduction to functional programming. Implementation of matrix algebra and information processing algorithms in a functional language style.