Dynamic Pipeline: An Adaptive Solution for Big Data

Julián Aráoz
Technical University of Catalonia–Barcelona Tech

Maria-Esther Vidal
Fraunhofer, IAIS, Germany

Edelmira Pasarella
Technical University of Catalonia–Barcelona Tech

Cristina Zoltan
Universidad Internacional de Ciencia y Tecnología – Panamá

ABSTRACT

The Dynamic Pipeline is a concurrent programming pattern amenable to be parallelized. Furthermore, the number of processing units used in the parallelization is adjusted to the size of the problem, and each processing unit uses a reduced memory footprint. Contrary to other approaches, the Dynamic Pipeline can be seen as a generalization of the (parallel) Divide and Conquer schema, where systems can be reconfigured depending on the particular instance of the problem to be solved. We claim that the Dynamic Pipelines is useful to deal with Big Data related problems. In particular, we have designed and implemented algorithms for computing graphs parameters as number of triangles, connected components, and maximal cliques, among others. Currently, we are focused on designing and implementing an efficient algorithm to evaluate conjunctive query.

KEYWORDS

Dynamic Pipeline; Parallelism; Big Data; Concurrency

Exploiting parallel architecture is a very difficult programing task. There are three basic models of parallelism: task, data, and pipeline parallelism [2]. The Dynamic Pipeline (DP) is a programming pattern that implements a concurrent program that can be easily parallelized. Informally, DP consists of stages connected by (logical) channels. Each stage receives values from upstream via inbound channels and performs some function on that data, usually producing and sending new values downstream via outbound channels. This is, stages can be seen as actors or filters. In a DP pattern, each stage has any number of inbound and outbound channels. The first stage is sometimes called the source or producer; the last stage is the sink or consumer.

Most systems that provide pipeline parallelism employ a construct-and-run model, i.e., programs do not take into account a particular instance of a problem. This is the case for MapReduce [3]. There are other approaches that construct a program using a problem instance, and then run the constructed program to solve the given problem, e.g., Spark [5]. In particular, Spark uses a synchronous computation model.

REFERENCES