MapReduce Program Synthesis

MohammadAmin Raeisi

Summary

This paper proposes a synthesizer which can automatically synthesize MapReduce-style distributed programs from input-output examples given by end users and it is implemented as $BIG\lambda$ tool. Their goal is to enable end users to specify large-scale data analyses through the simple interface of examples. They present a new algorithm and tool for synthesizing programs composed of efficient data-parallel operations that can execute on cloud computing infrastructure.

The MapReduce-style programming readily provides a common structure for the program: a composition of data-parallel operations, e.g., map followed by reduce, or a sequence of map and reduce. It restricts the space of possible programs and doesn't need searching for an arbitrary piece of code to realize the given input-output examples. In other words, the MapReduce paradigm provides us with a program *template* in which we need to fill the missing pieces. Capitalizing on this insight, they designed their program synthesis technique to be parameterized by a set of $higher-order\ sketches$ (HOS): templates that dictate a data-parallel structure on the synthesized program. For instance, the $map\ \Box\ .\ reduce\ \Box$ HOS instantiates the algorithm for finding a program composed of a *map* followed by a *reduce*.

To compute a solution of the synthesis task, their algorithm employs two cooperating phases, *synthesis* hand *composition*, that act as producers and consumers, respectively. (1) *Synthesis phase* (producers): In this phase, the algorithm infers the type of terms that may appear for each wildcard h in H which H is set of all possible HOSs. For instance, it may infer that needs to be replaced by a term of type $int \to int$. Thus, for each inferred type τ, the synthesis phase will produce terms of type τ. (2) *Composition phase* (consumers): For each HOS $h \in H$, the composition phase attempts to find a map μ, from wildcards to complete programs, such that μh is a solution of synthesis task. To construct the map μ, this phase *consumes* results produced by the synthesis phase.

Strengths

- The paper has presented the motivation and problem statement well by providing many good example in the introduction.
- It has shown that this method is better than the previous methods by presenting the diagrams.
- This is the first attempt to write a synthesizer for MapReduce in scala DSL.
- The MapReduce they have chosen for synthesis is very suitable because it is very effective in the field of big data today.
- The structure of MapReduce is well explained in part 2 by providing various examples, and those who are not familiar with it can follow the paper well.
- The text of the paper is clear and expressive.

Weaknesses

- It would be better to bring the pseudo-code of the algorithm in the paper to help better understand algorithms.
- It would be better to put the link to the implemented tool in the paper.
- It would be better to provide more diagrams and benchmarks to show the efficiency of their algorithm.
- In the introduction section, the generalities of the algorithm are not mentioned much.