Swift: A language for distributed parallel scripting

CS554 – Data Intensive Computing CWID–A20402686

Summary of the paper

This paper describes Swift's implicitly parallel and deterministic programming model, which applies external applications to file collections using a functional style that abstracts and simplifies distributed parallel execution. Scripting language Swift is made for converting application programs into parallel applications to be executed on multicore processors, clusters, grids, supercomputers and clouds. Issues of concurrent execution, composition and coordination of tasks are addressed by Swift. It uses C like syntax. There is no need for user to explicitly code parallel behavior and explicit transfer of file i.e. Swift is implicitly parallel and location-independent. Swift's function invocations are parallelized implies that functions produce same output for same input. In Swift, it is easy to call other scripts as small application. Swift makes use of concurrency allowed by data dependency within the same script and by other resources. Swift represents process as function in which input data file and parameters are function parameters and output data files are function return values. Swift uses construct of 'futures' which enables large-scale parallelism. Every Swift variable is a future. Swift executes external programs remotely. Data Element in Swift refers to every local variable, parameter, return and individual element of array and structure.

Variables of Swift functions can be declared global to make them accessible to every other function in the script. Primitives, mapped and collection are three basic classes of datatype. Integer, float, string and Boolean values are primitive types. Mapped types are used to declare data elements that refer to files external to the Swift script. The mapping process can map single variables to single files, and structures and arrays to collections of files. Mapper is associated with variable that is declared to be mapped file. It defines the file that is mapped to variable. Structure and Arrays are two collection types. Swift's built-in functions perform utility functions such as numeric conversion and string manipulation. Swift's application interface functions are declared by using 'app' keyword. They specify both interface and command line syntax. Swift's compound function invokes other functions and can contain flow-control statements. Arrays in Swift are declared by using the [] suffix. Swift uses foreach construct to obtain concurrency.

Swift also provides conditional execution through the if and switch statements and explicit sequential iteration through the iterate statement. Every data object in Swift contains three fields i.e. a value, a state, and a queue of function invocations. Swift data elements (atomic variables and array elements) are single assignment. Swift collection types (arrays and structures) are not single-assignment, but each of their elements is single-assignment. Swift provides built-in mapping primitives (mappers) that make a given variable name refer to a filename. These mappers can operate both as input mappers and as output mappers. To execute Swift program, it is invoked in its own working directory where program can find all its input files. At the time of exit, program leaves all files named by that application in the same working directory.

Applications of Swift are computational biochemical investigations, like protein structure prediction, molecular dynamic simulation and protein-RNA docking. Swift can also be used in discipline of analysis of satellite land-use imagery and investigation of molecular structure of glassy materials in chemistry. The performance of Swift is measured by synthetic benchmark results and application performance measurements.

How is this work different than the related work?

Work of this paper differs in many factors with many other systems. Coordination languages such as Linda, Strand, and PCN support the composition of implicitly parallel functions programmed in specific languages and linked with the systems. But, Swift coordinates the execution of distributed functions which are typically legacy applications which are coded in various programming languages, and can be executed on heterogeneous platforms. Linda uses primitives for concurrent manipulation of tuples, but Swift uses single assignment variables for coordination mechanism. The MapReduce programming model supports key-value pairs and two types of computation functions i.e. map and reduce. But, Swift provides a type system and allows the complex data structures and arbitrary computational procedures. Contrast to the Swift, MapReduce does not provide any mapping mechanism for input and output data. In terms of scheduling, MapReduce schedules computations within a cluster with a shared Google File System but, Swift schedules across distributed grid sites and deals with security and resource usage policy issues. FlumeJava builds on top of MapReduce primitives but Swift is abstract graph. Pegasus performs graph transformation with the knowledge of the whole workflow graph, but in Swift the structure of a workflow is constructed and expanded dynamically. In Dryad, graphs are explicitly developed by the programmer while in Swift, graphs are implicit. Dryad is used in clusters and single administrative domains, where Swift supports various platforms. Dryad tasks are written in C++, but Swift tasks can be written in any language. In GEL, user must define what is parallel and what is serial, but Swift states this based on data dependencies.

Identify the top 3 technical things this paper does well.

- **C like syntax -** Swift uses C like syntax
- **Remote Execution –** Swift executes external programs remotely and that too in parallel.
- **Reliability mechanism –** Swift provides three reliability mechanisms namely retries, restarts and replication.

Identify 3 things the paper could do better.

- Swift should have mechanism to minimize the cost in terms of space and internal object management while achieving automatic parallelization.
- There should be more reliable and efficient technique for avoiding job submission penalties.
- There should be support for efficient transitive queries that make some common queries expensive while querying indexed relation or XML database.

If you were to be an author of a follow up paper to this paper, what extensions would you make to improve on this paper?

If I was an author of this paper, I would have included the method to avoid job submission penalties.