SWIFT

Summary

This paper gives the detail explanation of SWIFT scripting language. SWIFT language is used for breaking up of single application programs into parallel applications and running them concurrently on multiple process, clusters, grids, clouds and supercomputer. Unlike any other scripting language SWIFT is capable of maintaining concurrent execution, composition and coordination of many independent computational tasks. Many parallel applications use Message Passing Interface(MPI) while others use coupling or orchestration of large number of parallel applications while SWIFT's primary focus is to provide minimal set of language constructs to specify how applications are glued together and executed in parallel at large scale. It standardizes and hides the processes and external data for distributed parallel execution of application programs. SWIFT is location independent and implicitly parallel and the user does not need to code explicitly for the synchronization or parallel behavior. SWIFT's functional model is deterministic if the called functions are deterministic. SWIFT can execute hundreds of thousands of programs invocations on distributed system and does not have the replicating capabilities like most of the scripting languages do (Perl, Python, shell), instead it calls the scripts as small application. It must be noted that the number of resources present on any shared resources varied with time, SWIFT makes use of maximal concurrency permitted by data dependencies within a script and by external resource availability.

The paper aims to make SWIFT a purely functional i.e. all operations have a well-defined set of inputs and outputs, all variables are write-once, and no script-level side effects are permitted by the language as functional programming allows consistent implementations of evaluation strategies, in contrast to the more common approach of eager evaluation. SWIFT allows you to define process architecture in such a way where input data files and process parameters become can be used as function parameters. SWIFT also provides a high-level representation of collections of data (used as function inputs and outputs) and a specification ("mapper") that allows those collections to be processed by external program. SWIFT is based on synchronization construct of futures which can be used in large scale distributed systems. Using futures based evaluation idea allows for automatic parallelization without need for dependency analysis. SWIFT gives the same benefit to scripting as RPC gives to programming languages and transparency to remote application execution.

This paper further goes into the detail study of the performance results which on SWIFT which was conducted at Argonne National Laboratory. SWIFT was used to submit 2,000 tasks to a 16-core x86-based Linux compute server. In the first test, which ran for 5 seconds, it was observed SWIFT can sustain 100 concurrent applications executions at a CPU utilization of 90%, and 200 concurrent executions at a utilization of 85%. In the second test which ran for 10 seconds to Intrepid, the 40,000-node IBM Blue Gene/P system with 20,480 tasks, it was observed that for tasks of 100 seconds, SWIFT achieves a 95% CPU utilization of 2,048 compute node.

How is this work different than the related work

There are many coordination languages such as Linda, Strand and PCN which support composition of implicitly parallel functions programmed in specific languages and linked with the systems, whereas SWIFT supports the coordination and execution of legacy applications which are coded in different programming languages on heterogeneous platform. Linda uses primitives for concurrent manipulation of tuples via a shared "tuple space". SWIFT uses single-assignment variables much like Strand and PCN. Linda, Strand, PCN and SWIFT are all data flow driven i.e. process executes when data is available. Mapreduce is also used for programing of large scale datasets. It provides two function map and reduce borrowed form functional programming. Map function iterates over the set of items, perform operations and generates new set of items, on the other hand reduce function aggregate all the items present in the datasets. The

input data gets partitioned by runtime system and programs are scheduled over the large commodity machines in large cluster. Sawzlla is built on Mapreduce and supports filtering and aggregation.

Goals of Mapreduce and SWIFT are same but Mapreduce provides key-value pairs as input or output datasets and two types of computation functions, map and reduce. In MapReduce, input and output data can be of several different formats, and new data sources can be defined. While SWIFT provides a type system and defines the complex data structure and arbitrary computational procedures. SWIFT enables a more flexible mapping mechanism to map between logical data structures and various physical representations. SWIFT does not automatically partition input datasets as MapReduce does. Datasets in SWIFT can be organized in structures, individual items in a dataset can be transferred accordingly along with computations. MapReduce schedules computations within a cluster with a shared Google File System; SWIFT schedules across distributed grid sites.

FlumeJava is like Swift in concept but it is built on the top of Mapreduce primitive instead of abstract graphs as in SWIFT. BPEL is a Web service-based standard which regulates how a set of Web services interact to form a larger, composite Web service BPEL can transfer data via XML but does not provide support for iteration of datasets. It also does not provide any support for logical XML view and arbitrary physical representations. DAGMan provides a workflow engine that manages Condor jobs organized as directed acyclic graphs (DAGs) in which each edge corresponds to an explicit task precedence. It lacks dynamic features like iteration or conditional execution. Pegasus is primarily a set of DAG transformers. It performs a graph transformation with the knowledge of whole workflow graph whereas in SWIFT the structure of workflow is built and expanded dynamically. BASH allows a user to define a dataflow graph but execute on single cluster or parallel system. GEL defines an order in which a program should run but lacks runtime sophistication and platform support. Dryad infrastructure can be used for running data-parallel programs on a parallel or distributed system but its tasks are written in C++ whereas SWIFT tasks can be written in any language.

The top 3 technical things this paper does well

- Paper has explained the fundamental concepts of SWIFT, its implementation and performance characteristics on Blue Gene/P supercomputer and tasks executed over a time period.
- Paper has successfully managed to compare the workflow of different scripting languages like Mapreduce, Linda, PCN etc. with SWIFT and made some strong observational points.
- Paper has defined the SWIFT language and its components and functions in detail i.e. data Models and Types, SWIFT mappers, Arrays and parallel execution etc.

3 things the paper could do better

- Paper could have provided experiments results of number of task execution of different scripting languages with SWIFT on cluster, grid etc.
- Paper could have provided and discussed the cost of I/O when application runs on parallel system as well as on remote location.
- Paper could have discussed the file transfer optimization strategy in distributed system paradigm.

Extensions would you make to improve on this paper

Paper has significantly proved and mentioned the task execution on thousands of cores but I would have also measured the performance of tasks execution on millions of cores. As we have seen in many cases that as the number of cores increases the application parallelism becomes more complex to handle and

the system may fallout or fail to perform on that many cores. I would have also included File System access optimization for many users when application runs on HPC systems.