DEMO: CIRCE – A runtime scheduler for DAG-based dispersed computing

Aleksandra Knezevic, Quynh Nguyen, Jason A. Tran, Pradipta Ghosh, Pranav Sakulkar, Bhaskar Krishnamachari and Murali Annavaram
[aleksank,quynhngu,jasontra,pradiptg,sakulkar,bkrishna,annavara]@usc.edu
University of Southern California
Los Angeles, CA 90007, USA

ABSTRACT

CIRCE (CentralIzed Runtime sChedulEr) is a runtime scheduling software tool for dispersed computing. It can deploy pipelined computations described in the form of a Directed Acyclic Graph (DAG) on multiple geographically dispersed compute nodes at the edge and in the cloud. A key innovation in this scheduler compared to prior work is the incorporation of a run-time network profiler which accounts for the network performance among nodes when scheduling. This demo will show an implementation of CIRCE deployed on a testbed of tens of nodes, from both an edge computing testbed and a geographically distributed cloud, with real-time evaluation of the task processing performance of different scheduling algorithms.

ACM Reference format:
Aleksandra Knezevic, Quynh Nguyen, Jason A. Tran, Pradipta Ghosh, Pranav Sakulkar, Bhaskar Krishnamachari and Murali Annavaram. 2017. DEMO: CIRCE – A runtime scheduler for DAG-based dispersed computing. In *Proceedings of SEC '17, San Jose / Silicon Valley, CA, USA, October 12–14, 2017,* 2 pages.
DOI: 10.1145/3132211.3132451

1 INTRODUCTION

In recent years, there has been a rapid growth of computationally intensive applications, such as image and voice recognition, on end user devices like cellphones. In the internet of things (IoT) era, with the boom of low cost microcomputers, there is growing interest to perform a part or all of the required computations on the edge devices rather than sending it to a central cloud, thereby avoiding network overhead and delays. The field of dispersed computing is focused on using all the available computing resources dispersed throughout the path from edge devices to the cloud to perform a more timely and optimized processing of data in terms of computation cost and network overhead. To this end, our research is focused on the development of a dispersed computing platform for distributing the execution of networked applications that can be described in the form of a Directed Acyclic Graph (DAG) based task graph.

In this demo, we present our recently developed runtime scheduling software tool for dispersed computing, which we refer to as CentralIzed Runtime sChedulEr (CIRCE). CIRCE takes a task DAG as an input and assigns each task to a node from a geographically distributed cluster. Whenever a data to be processed is received, it is dispatched by CIRCE to the corresponding input nodes of the DAG. On the other hand, whenever the pipelined processing of the data is complete, the last node of the task graph outputs the data to the CIRCE node. CIRCE also implements a run-time network profiler that monitors the pairwise traffic and the communication delay performance between the worker nodes to incorporate a timely and accurate estimation of network overhead cost into the scheduling.

SEC '17, San Jose / Silicon Valley, CA, USA
 978-1-4503-5087-7/17/10…$15.00
DOI: 10.1145/3132211.3132451

2 DESCRIPTION OF CIRCE

CIRCE consists of several tools used for profiling, static scheduling, and run-time scheduling. To use CIRCE, the set of computing nodes need to include a master node and a set of worker nodes, and each task of the input task DAG is assumed to be implemented in a separate Python script as a function. CIRCE runs in several phases as in Figure 1. Next, we explain different components of the CIRCE and how they interconnect.

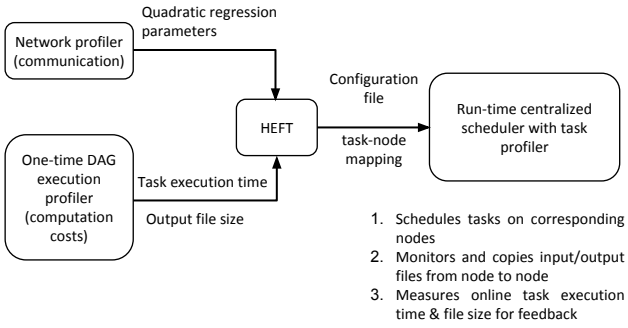

Figure 1: CIRCE System components

2.1 Profiling

In our proposed CIRCE architecture, each worker node of our dispersed computing testbed runs a network profiler and one time DAG execution profiler of the target DAG before any task scheduling. Once the information is collected and reported back to the master node, the scheduler leverages this information to schedule each task to specific worker nodes.

2.1.1 Network profiler. This tool automatically schedules and logs communication information of all links between nodes in

the network, which gives the quadratic regression parameters of each link representing the corresponding communication cost. The quadratic function represents how the file transfer time depends on the file size (based on our empirical finding that a quadratic function is a good fit). The quadratic regression parameters are sent back to the master node and stored in the MongoDB server on the master node, which is used by the HEFT algorithms described in the next section.

2.1.2 One-time DAG execution profiler. This tool runs the whole DAG on each worker node and measures the execution time of each task and the size of the output data it passes to its child tasks. The results are saved in a text file, which is sent back to the master node. These files are later used by HEFT. For easier and modular deployment of the One-time DAG execution profiler on every node of our dispersed computing testbed setup, this profiler is set up to run inside a Docker container on each node.

2.2 Scheduling Algorithm

Once the information about the DAG and its computational and data requirements are determined, a centralized scheduling algorithm is run to determine which task to run on which compute node. As a starting point, we identified an open source implementation [2] of the classic Heterogeneous Earliest Finish Time (HEFT) algorithm [1] for scheduling a DAG on distributed computers and adapted it to work with CIRCE. HEFT uses the data obtained from the Network profiler and the One-time DAG execution profiler to produce a **configuration file** that describes the mapping of tasks onto the available worker nodes. We are working to incorporate other scheduling algorithms into the CIRCE framework. Other researchers will be able to use the platform to evaluate their own algorithms as well.

2.3 Run-time scheduler with task profiler

Once a **configuration file** with task to node mapping is generated, the run-time scheduler is executed centrally on the master node. The tool then copies the task files to each of the corresponding worker nodes and start the processes to monitor the input and output folders. Input files are received on the master node, and upon the detection, they are transmitted to the node executing the first task. When the file is received on a worker node, the task is executed, and the output file is sent to the nodes where worker node's child tasks are scheduled. Output files of the final task are sent back to the master node. The built-in task profiler running side by side with the centralized scheduler measures the online execution time of each scheduled task and the size of the output data passed to its child tasks. The results of the task profiler are stored in text files, sent back to the master node, and stored in a MongoDB database server on the master node.

3 OVERVIEW OF THE DEMO

The proposed demo will show how a DAG based application pipeline can be scheduled and executed in a distributed manner on heterogeneous cloud and edge nodes.

The 9 task DAG, shown in Figure 2, is an example of a simple application where each task reads a text file, does some processing of the text and writes an output file. In our testing so far, we have evaluated CIRCE on a 10-node geographically dispersed cloud network formed on DigitalOcean, with one acting as the master node. Figure 2 also shows the node (near the task circle) where the corresponding task is scheduled by HEFT. During the demo we will show CIRCE working on a heterogeneous system consisting of a raspberry pi based edge computing cluster as well as dispersed cloud-based nodes.

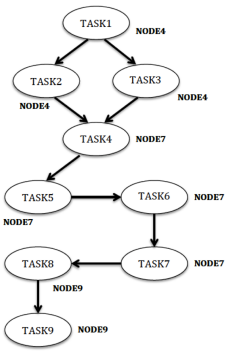

Figure 2: An illustration of a task DAG

The first step in running CIRCE is to run the scheduler script on the master node. It copies required files to the worker nodes and start monitoring the input folder. In our current implementation it is assumed that processing begins when input files arrive to the input folder on a master node. They will be detected and copied to the worker node where the corresponding input processing task is scheduled. The files will go through the nodes, as scheduled, and the final output files appear in the output folder on the master node.

The demo will showcase a comparison between different scheduling algorithms such as a baseline scheduler, HEFT [1, 2], and others, in terms of their task execution and data communication latencies. We also plan to describe our ongoing work on developing novel schedulers for optimized pipelined execution, on developing run-time adaptation to handle network and node dynamics, and incorporate state of the art container orchestration frameworks such as Kubernetes and Apache Mesos.

ACKNOWLEDGMENTS

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0 053. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] H. Topcuoglu, S. Hariri, M.Y. Wu, Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing, IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 3, pp. 260 - 274, 2002.
[2] Ouyang Liduo, HEFT Implementation Original Source Code, https://github.com/oyld/heft.