WAVE: A Distributed Scheduling Framework for Dispersed Computing

Pranav Sakulkar, Pradipta Ghosh, Aleksandra Knezevic, Jiatong Wang, Quynh Nguyen, Jason Tran, H.V. Krishna Giri Narra, Zhifeng Lin, Songze Li, Ming Yu, Bhaskar Krishnamachari, Salman Avestimehr, Murali Annavaram

v0: December 29, 2017, v1: January 15, 2018

1. Introduction:

Given a task graph that is represented as directed acyclic graph (DAG) and a network of network compute points (NCPs), the dispersed computing scheduler needs to figure out a mapping from the tasks to the NCPs with the goal minimizing the average end-to-end latency of the incoming data-frames. The scheduler needs to know the computing resources availability at each of the NCPs and the qualities of the links connecting any two devices in order to come up with a mapping. The knowledge of the compute resources is important, since different tasks might be executed on different NCPs. In such a case, the link quality knowledge is especially important as the output of a task being executed at an NCP will need to be shipped to its children task being executed at another NCP.

In a centralized schedulers, a central master node needs to know the compute profiles of all the NCPs and the link qualities for all the links. This assumption is not particularly suited for dispersed computing applications. Especially as the number of compute nodes grows, the number of message exchanges required grows quadratically. Also, there may not be direct links from every node to the master node and their messages would need to be relayed by some other nodes. In spite of these difficulties, previous work such as [2] has focused solely on centralized schedulers.

In this report, we consider the case where the scheduling is done in a distributed manner by multiple collaborating nodes located at different geographical locations. In such cases, each scheduling node only needs to know about its neighbours, their compute profiles and the link qualities to them. We implement the distributed scheduler in Python and deploy it on our Kubernetes cluster of 100 nodes. The schedule determined by our scheduler is used to deploy the dispersed computing application using the framework CIRCE [1].

The architecture of the dispersed computing testbed is explained in figure 1. We start with the application DAG, which is used by the computation profiler that profiles the NCPs for their computing resource availability. The network profiler profiles the links connecting every pair of NCPs in a distributed manner. This information is the used by the scheduling algorithm HEFT (a centralized algorithm) and WAVE (a distributed algorithm) to come up with a mapping from the tasks of the task graph to the NCPs. CIRCE, our dispersed computing deployment framework, uses this mapping and deploys the tasks over the NCPs. CIRCE also performs run-time profiling of the task executions over time and logs them centrally, which can then be used for analyzing the schedules.

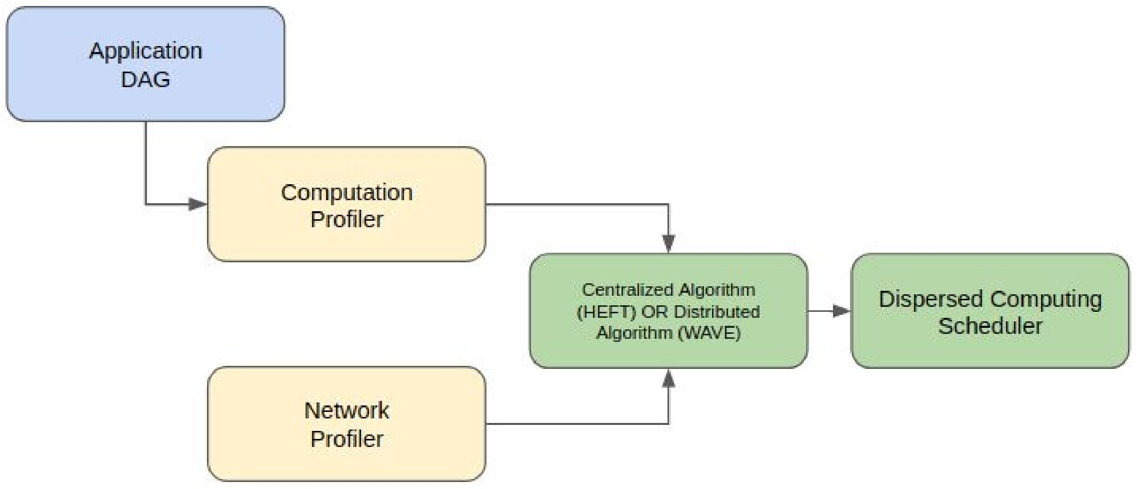
Application DAG
Computation Profiler
Network Profiler
Centralized Algorithm (HEFT) OR Distributed Algorithm (WAVE)
Dispersed Computing Scheduler

Figure 1. Architecture of the dispersed computing pipeline.

2. WAVE: A Distributed Scheduler

Our scheduler is initialized by the WAVE master node, which has the information about the task graph. Based on the location of the input data, it determines the NCPs for each of the input tasks in the DAG. Also the master node determines a unique parent controller for each task in te task graph using following routine 1 shown below. At the WAVE master node, following routine is executed to determine the controllers for each task.

Routine 1: Controller section routine

1) Iterate over tasks of the the task graph in their topological orders.
2) For each non-input task, check if any of its parent tasks are already controllers.
3) If only one of the parents is already a controller, then appoint that parent as the controller for this task.
4) If no parent is already a controller, then choose the task with smaller topological index as the parent.

The task graph, input NCPs and parent controllers for each task are then sent to all NCPs. All NCPs are waiting to receive their task assignments. Whenever an NCP hears its task responsibility, it first needs to check if the assigned task is also a controller for any of the other tasks. The NCP can check this by looking up the parent controller information it has received from the WAVE master. If the NCP is a controller parent for some tasks of the task graph, it needs to do a NCP assignment for the children tasks. It runs following routine 2 to perform the child appointment.

Routine 2: Scheduling algorithm at the controller

1) Iterate over the children tasks in their topological orders.
2) For each task, randomly select an NCP from the neighbouring nodes.

3) Convey the task appointments to the selected NCPs and the WAVE master node.

This is how the NCPs get to know the tasks assigned to them by the controlling parent tasks. When the WAVE master hears about all the task assignments, it starts the CIRCE deployment framework by providing the task graph and the mapping of these tasks to the compute nodes.

Note that the step (2) in routine 2 selects the NCPs randomly, however our system implementation is very modular and we are working on replacing it with a child appointment algorithm that considers the execution profiles of the neighbours, the link qualities connecting them and also the computing and communication requirements of the concerned tasks.

3. Network Anomaly Detection DAG:
For evaluation of our dispersed computing framework, we consider a network anomaly detection application, where the goal is to monitor the traffic flowing in a network and detect any network flow anomalies. An observation point, such as a router, monitors the packets and logs some information from the headers like the source IP address, the source port number, the destination IP address and the destination port number. This log is the used by the task graph to detect the anomalies in the network traffic. The source code for a version of this DAG is available online [4].

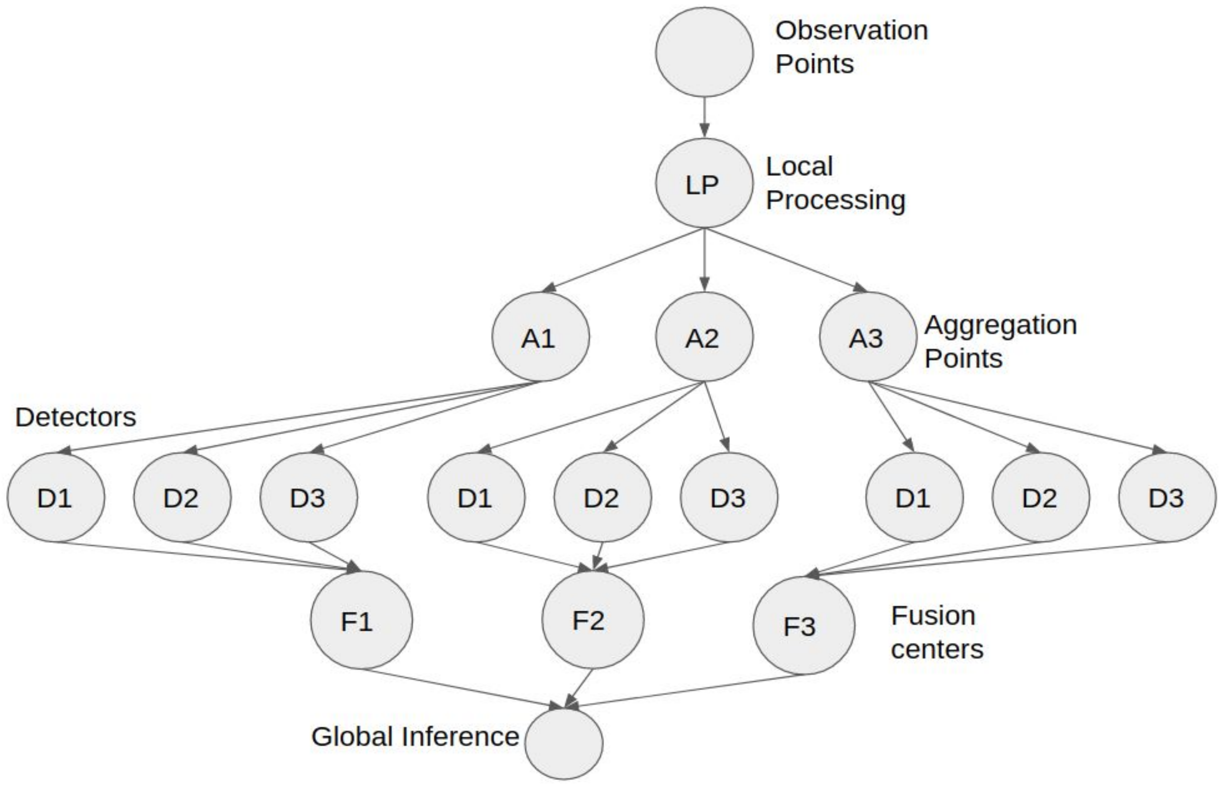

Figure 2. DAG for the Application

1. **Local Processing (LP):** The local processing unit hashes the traffic using the IP address. The traffic is then split into 3 independent streams based on the evaluated hash value and sent to 3 different aggregation points as shown in figure 2.

2. **Aggregation Point (AP):** In this example, we have considered only one observation point. However, in general, there can be several observation point and an aggregation point is needed to merge the traffic coming from several sources. The aggregated traffic is then sent to the detectors.

3. **Detectors:** Any network anomaly detector is bound to have false alarms and missed detections. Using several detectors in parallel and combining their outputs together is intended to provide more robustness to the detection performance. We use four different detectors on each data-stream: simple detector, Astute detector, DFT-based detector, TeraSort-based detector. It must be noted that the DFT-based and the TeraSort-based detectors use coded computing and currently are run on 4 different nodes, where there are one master and 3 worker nodes for each of the detectors.

4. **Fusion Point (F):** Fusion center combines the detected anomalies from different detectors together. This indicates the detected anomalies for the particular data-stream.

5. **Global Inference (G):** The global inference unit combines the detected anomalies for each stream into one global inference about the observed traffic.

4. Experimental Setup:

In order to test and evaluate our novel framework, we have created a 100-node cluster of geographically separated droplets on DigitalOcean. We use droplet with 8GB RAM and 80GB storage as our master node for the Kubernetes cluster. This the central node that orchestrates the execution of all the different components of our framework. The worker nodes have 2 different profiles: 2GB RAM with 40GB storage and 3GB RAM with 20GB storage. We run different containers on these nodes for each of the components of our system. Since we need to profile all the nodes and the links, there is one profiler container running on each of the nodes. There is also one WAVE container on each of the nodes. Some of these WAVE containers will become controllers when the task assignments take place in a distributed manner. Finally when the tasks assignment is completed, CIRCE deploys its own containers on the nodes that have been assigned some tasks.

5. Results:

We compare two different types of scheduling algorithm: centralized algorithm and WAVE (decentralized algorithm). Note that the decentralized algorithm is currently naive, doing random child appointments (this will be improved in the future). We test our system twice with 5 sequential input files provided in each test.

The average end-to-end execution times for the 2 tests are provided here:

	Centralized Algorithm	WAVE
Test 1	126.61 sec	135.82 sec
Test 2	127.59 sec	136.37 sec

As we can see, the mapping provided by the centralized takes 9.01 seconds fewer than that coming from the WAVE algorithm.

6. Conclusion:

In our work, we have developed a 100-node Kubernetes based cluster that integrates into a system, 3 components: a profiling module that monitors the resources and network links, a scheduler module that can use the profiled information to come up with a mapping of the tasks to NCPs and CIRCE, a deployment module that implements the mapping over the NCPs.

A centralized algorithm that assumes the availability of all profiling information locally performs better than WAVE, our decentralized algorithm. It must be noted that the distributed algorithms in general can only be as good as the centralized algorithms. In our case, the gap between these allocations can be bridged by using the profiling information of the neighbours in making the child appointment decisions in WAVE.

Acknowledgement

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053. 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] Aleksandra Knezevic, Quynh Nguyen, Jason A. Tran, Pradipta Ghosh, Pranav Sakulkar, Bhaskar Krishnamachari, and Murali Annavaram, “DEMO: CIRCE – A runtime scheduler for DAG-based dispersed computing,” The Second ACM/IEEE Symposium on Edge Computing (SEC) 2017.

[2] Topcuoglu, Haluk, Salim Hariri, and Min-you Wu. "Performance-effective and low-complexity task scheduling for heterogeneous computing." IEEE transactions on parallel and distributed systems 13.3 (2002): 260-274.

[3] Fontugne, Romain, Johan Mazel, and Kensuke Fukuda. "Hashdoop: A mapreduce framework for network anomaly detection." Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on. IEEE, 2014.

[4] Open Source code release: Distributed Network Anomaly Detection DAG for Dispersed Computing

https://github.com/ANRGUSC/DNAD