On Cost Efficient Dataflow Computing Program Deployment in SDN Managed Distributed Computing Environment

Yating Zhang[1], Yuepeng Li[2], Long Zheng[3](✉), and Deze Zeng[2]

[1] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[2] School of Computer Science, China University of Geosciences, Wuhan, China
[3] College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China
lzheng.aizu@gmail.com

Abstract. Dataflow computing has been regarded one of the most promising computing paradigms in the big data era. With the vast distribution of data sources, it is significant to deploy the dataflow based applications in distributed environment to digest these data. In dataflow computing, the data flows shall be transferred between different processing units to the accomplish the predefined semantics. Software-defined networking (SDN) has emerged as an effective network management technology to orchestrate the data flows among these processing units. For each data flow, a forwarding rule shall be inserted into the forwarding table of each switch on the routing path. However, the number of rules that can be inserted in one forwarding table is limited. We are motivated to take such constraints into the consideration of dataflow applications deployment in distributed computing environment managed by SDN. An efficient deployment algrotithm is proposed and evaluated in this paper.

1 Introduction

Dataflow computing, thanks to its charming characteristics in dealing with the large and stream data, has raised lots of attention for embracing the upcoming big data era. Specially, with the fast development of cloud computing, it has been widely regarded that it is ideal to deploy the dataflow computing programs in distributed environments to explore the vast cloud computing resources. As a result, many distributed dataflow computing based frameworks, e.g., GraphX [10], CIEL [6], TensorFlow [1], etc., have been proposed. These frameworks all share the same design principle that the program can be described as a directed graph where the data flows between the processes according to the predefined semantics. This raises one natural question on how to deploy these processes in the distributed computing environment, as illustrated in Fig. 1. This is generally referred as virtual network embedding (VNE), which has attracted much attention in the literature, as surveyed in [4]. However, we notice that existing network embedding studies usually consider traditional network architecture.

I. Romdhani et al. (Eds.): CollaborateCom 2017, LNICST 252, pp. 27–36, 2018.
https://doi.org/10.1007/978-3-030-00916-8_3

Software-defined networking (SDN) [9] has been regarded as one of the essential technologies in the next-generation networks by allowing centralized management of the data flows in the network. Although it is promising to apply SDN to orchestrate the data flows between different processes, a new challenging problem due to the inherent characteristics of SDN is introduced. That is, SDN relies on the forwarding rules written in the SDN switches to manage forwarding behaviors for different flows passing through the switches. The rules are usually written in Ternary Content Addressable Memory (TCAM) such that the rules can be read in parallel for fast entry matching and corresponding processing. However, TCAM is expensive and power hungry. This limits the size of the forwarding table, or rule space.

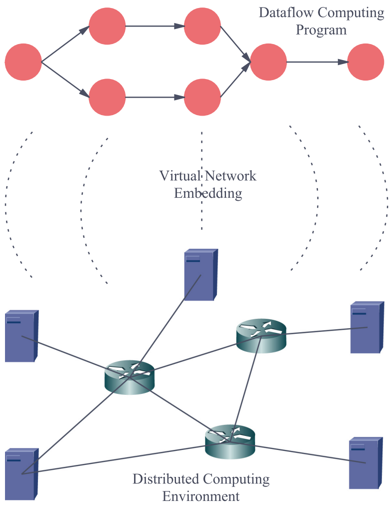
Dataflow Computing Program
Virtual Network Embedding
Distributed Computing Environment

Fig. 1. Illustration of embedding a dataflow computing program into an SDN managed distributed computing environment

Therefore, it is significant to re-investigate the network embedding problem in SDN managed distributed environment, especially with the consideration of the inherent characteristics of SDN itself, e.g., the forwarding table size limitations. In this paper, we are motivated to investigate the cost-efficient dataflow application deployment problem in SDN managed distributed computing environment. The main contributions of this paper exist in the following folds:

– To our best knowledge, we are the first to take the SDN characteristics in the consideration of VNE. In particular, we consider the problem of cost-efficient dataflow application deployment in SDN managed distributed computing environment.
– We formulate the cost-efficient network embedding problem into a mixed integer linear programing (MILP) form and further propose a relaxation-based polynomial heuristic algorithm.

– Through extensive simulation based studies, the high efficiency of our proposed heuristic algorithm is proved by the fact that it performs much close to the optimal solution.

The rest of the paper is organized as follows: existing related work is summarized in Sect. 2. The system model is given in Sect. 3. Section 4 presents our MILP problem formulation. Section 5 proposes the heuristic algorithm. Performance evaluation results are reported in Sect. 6. Finally, Sect. 7 concludes this paper.

2 Related Work

To explore the vast distributed cloud computing resources, much effort has been devoted to study of how to deploy a dataflow computing program in distributed computing environment with different goal. For example, Chowdhury et al. [3] address the efficient mapping of virtual nodes and virtual links onto the substrate network resources to increase revenue and decrease cost of the infrastructure provider in the long run, in addition to balancing load of the substrate network resources. Sun et al. [11] consider the case that the user demands and the corresponding virtual network requests change dynamically, and study on how to reconfigure an existing embedding to minimize the reconfiguration cost. Botero et al. [2] consider an energy aware VNE with the goal of providing optimal energy efficient embeddings. After comprehensive survey on existing studies on VNE in the literature, we noticed that all existing studies consider traditional network architecture and none of them takes SDN into consideration.

The functionality of SDN deeply relies on the forwarding rule deployed in the SDN switches. While, as we have known, the forwarding rule space on the SDN switches is usually size-limited due to the expensive and power-hungry TCAM. As a result, pioneering researchers have widely discussed how to efficiently manage the SDN rules in the size-limited rule spaces. For example, Giroire et al. [5] focus on applying SDN for energy aware routing with respecting capacity constraints on links and rule space constraints on SDN routers on backbone networks. Kang et al. [8] present efficient rule-placement algorithms to deploy predefined forwarding policies in general SDN switches while preserving the rule-space constraints. Although the rule space constraints have already been widely discussed in the literature, it can be noticed that existing studies usually assume end-to-end traffic model. When it comes to dataflow computing paradigm, there are a number of dependent communication nodes. Existing modeling and optimization algorithms cannot be applied directly.

3 System Model

We consider a substrate network consisting of a number of geographically distributed servers interconnected by SDN switches. Hence, we can denote the substrate network as an undirected weighted graph $G^s = (N, S, L)$, where N

and S denote the set of servers and the set of SDN switches, respectively, and L is the set of communication links between the servers and the switches, as well as the ones between the switches. For each substrate server $n \in N$, the amount of available resource capacity is denoted as C_n. For each SDN switch $s \in S$, the forwarding table capacity (i.e., rule space size) is denoted as C_s. For each substrate network link $l \in L$, the amount of available communication resource capacity is denoted as C_l.

A dataflow computing program to be deployed can be abstracted as a virtual network request. As a result, we similarly model a dataflow program as a weighted directed acyclic graph (DAG) $G^v = (P, E)$, where P is the set of constituent processes of the dataflow computing program and E denotes the data dependency relationships between these processes. If there is an edge e_{uv} from process $u \in P$ to process $v \in P$, it indicates that the output flow of u shall flow to v as input flow. In other words, u is the parent process of v and v is the child process of u. A process may generate multiple output flows to different child processes. Different output flows may have different flow volume expansion relationship, either increase or decease. We therefore define $\alpha_{uv}, u, v \in P$ to denote the ratio of stream volume on edge $e_{uv} \in E$ to the total input stream volume to u. For each process $v \in V$, certain units of resources are needed. We denote the resource requirement of process $p \in P$ as R_p. Without loss of generality, we specially define p_0 as the first process as the front-end proxy that each computing request shall visit. Similarly, we define p_1 as the child process of p_0. The proxy process p_0 has been deployed on all the servers such that each can receive the computing requests from the clients. We view each request as a flow with different rates that shall traverse all the constituent processes in the dataflow computing program. We denote the set of flows as F.

4 Problem Formulation

Extended Dataflow Computing Graph Construction: We notice Gu et al. [7] invents an extended graph for VNE, with the joint consideration of substrate network and virtual network. Following their work, we extend our dataflow computing DAG into an extended dataflow computing graph $G^e = (V^e, S, E^e)$ by integrating the substrate network architecture as follows. For each vertex $v \in V$, we replicate it into $|N|$ copies in G^e such that each vertex $v \in V^e$ represents one possible process placement. As a result, V^e has $|V| \cdot |N|$ vertices, i.e., $|V^e| = |V| \cdot |N|$. The set S is inherited from substrate network graph G^s. Next, we construct the edge set E^e between the vertices in V^e and the switches in S. Let us first define $h(v)$ and $t(v)$ denote its hosting server and corresponding process for vertex $v \in V^e$, respectively. Specially, for tractability, we define $h(s) = s, \forall s \in S$. Then, for any vertex $v \in V^e$, if there is a link from $h(v_1) \in V^s$ to $s \in S$ in G^s, we create an edge from v to s in G^e, i.e., adding an edge e_{vs} to E^e, whose unit cost is inherited from G^s as $C_{h(v)s}$. Such process repeated until all vertices in the extended dataflow computing graph G^e.

After constructing the extended graph G^e, we start to build a formal optimization model to describe the problem to be studied as follows.

Process Placement Constraints: To ensure the functionality of the dataflow computing program, the first requirement is that all the processes must be completed placed in the substrate network. To this end, we first define a binary x_v to indicate whether virtual vertex v is chosen ($x_v = 1$) or not ($x_v = 0$). According to our extended graph construction process, it can be seen that $x_v = 1$ indicates that process $t(v)$ is placed on server $h(v)$. For each process, there must be one instance in the substrate network. This is equivalent to

$$\sum_{v \in V^e, t(v)=p} x_v = 1, \forall p \in P. \tag{1}$$

From the perspective of the substrate server, the processes that can be placed on it is limited by its resource capacity, i.e.,

$$\sum_{v \in V, h(v)=n} x_v R_{t(v)} \leq C_n, n \in N. \tag{2}$$

Flow Distribution Constraints: Let $\lambda_{uv}^{f_{p'p}}$ denote the flow rate between any two vertices u, v for flow $f \in F$ going from process p' to p, if there exists an edge $e_{uv}, e_{uv} \in E^e$ in graph G^e. If the rate of a flow going through v is larger than 0, it indicates that process $t(v)$ must be placed in server $n(v)$, i.e., $x_v = 1$. Therefore, we have

$$\frac{\sum_{f \in F} \sum_{e_{uv} \in E^e} \lambda_{uv}^{f_{p't(v)}}}{A} \leq x_v \leq A \sum_{f \in F} \sum_{e_{uv} \in E^e} \lambda_{uv}^{f_{p't(v)}}, \forall v \in V^e, e_{p't(v)} \in E, \tag{3}$$

where A is an arbitrary large number.

For each process $p \in P$, it receives one or several flows from its parent processes p' and generates one or several output flows, according to the predefined dataflow computing semantics. Such relationship can be described using the flow conservation constraints on the vertices in the extended graph as

$$\alpha_{pp''} \sum_{\substack{t(v)=p \\ e_{uv} \in E^e}} \lambda_{uv}^{f_{p'p}} = \sum_{\substack{t(v)=p \\ e_{vw} \in E^e}} \lambda_{vw}^{f_{pp''}}, \forall f \in F, e_{p'p}, e_{pp''} \in E. \tag{4}$$

Note that we treat the front portal as the first process that each flow shall visit. Therefore, the input flow rate of each front portal process, i.e., p_0, has already been determined. That is,

$$\Lambda_v^f = \sum_{\substack{t(v)=p_0 \\ e_{vw} \in E^e}} \lambda_{vw}^{f_{p_0 p_1}}, \forall f \in F, v \in V^e. \tag{5}$$

For each switch $s \in V^e$ in the extended graph, unlike the process which may make some data processing, it simply forward the flow to either another switch or a server deployed with wanted process. Therefore, we have:

$$\sum_{e_{us} \in E^e} \lambda_{us}^{f_{pp'}} = \sum_{\substack{e_{sw} \in E^e \\ t(w)=p' \text{ or } w \in S}} \lambda_{sw}^{f_{pp'}}, \forall s \in S, e_{pp'} \in E. \tag{6}$$

As each link is with certain communication capacity, the total stream volume shall not exceed its capacity. That is,

$$\sum_{e_{pp'} \in E} \sum_{\substack{e_{uv} \in E^e \\ h(u)=x \\ h(v)=y}} \lambda_{uv}^{f_{pp'}} \leq C_{l_{xy}}, \forall l_{xy} \in L. \tag{7}$$

Rule Space Constraints: As all the servers are inter-connected by SDN switches, the flows between the dependent processes deployed on different servers shall go through the corresponding switches. Each flow shall occupy one rule on each switch along the routing path. Therefore, for each flow $f \in F$, whenever there is a non-zero flow going through a switch $s \in S$, it indicates that there is one rule in the forwarding table of switch s. To this end, we first define binary variable to describe whether flow f going through s ($x_s^f = 1$) or not ($x_s^f = 0$), which can be equivalently described as a linear function of the flow as

$$\frac{\sum_{e_{us} \in E^e} \lambda_{us}^{f_{pp'}}}{A} \leq x_s^f \leq A \sum_{e_{us} \in E^e} \lambda_{us}^{f_{pp'}}, \forall s \in S, f \in F, e_{pp'} \in E, \tag{8}$$

where A is an arbitrary large number.

For all the flows, the total rule space needed in each switch shall not exceed its rule space capacity. That is,

$$\sum_{f \in F} x_s^f \leq C_s, \forall s \in S. \tag{9}$$

A Joint MILP Formulation: We are interested in minimizing the total communication cost for all the dataflow computing requests as

$$Cost_{com} = \sum_{f \in F} \sum_{e_{uv} \in E^e} \lambda_{uv}^{f_{t(u)t(v)}} \cdot P_{h(u)h(v)}. \tag{10}$$

By summing up all above, we get the following **Com-Min** problem:

$$\begin{aligned} \textbf{Com-Min:} \quad \min: & \sum_{f \in F} \sum_{e_{uv} \in E^e} \lambda_{uv}^{f_{t(u)t(v)}} \cdot P_{h(u)h(v)}, \\ \text{s.t.}: & \ (1)-(9) \\ & x_v \in \{0,1\}, \forall v \in V^e, x_s^f \in \{0,1\}, \forall f \in F, s \in S. \end{aligned}$$

Note that it is computationally prohibitive to solve this MILP problem due to the involvement of integer variables x_v, especially in large-scale network cases. To tackle this problem, we design a low-complexity heuristic algorithm in the next section.

5 Heuristic Algorithm Design

Relaxation has been widely regarded as an efficient way to address the MILP problem. Therefore, we are motivated to propose a relaxation-based heuristic algorithm by relaxing all the binary variables. Following such principle, we design our relaxation-based algorithm shown in Algorithm 1.

As shown in line 1 of Algorithm 1, we first relax all the binary variables into real ones ranging in $[0, 1]$ to obtain a linear programing (LP) model **Com-Min-LP**, which can be solved in polynomial time. Once the values of $x_v, \forall v \in V^e$ are determined, the only thing left is to schedule the flows towards minimal cost under the constraints of SDN switches' rule space size limitations. In line 14, we first take the values of $x_v, \forall v \in V^e$ into **Com-Min-LP** ans solve the resulted in LP formulation to obtain an initial flow scheduling. Note that, we as relax the variables x_s^f into real number variables, the rule space constraints may be violated. Therefore, starting from 15, we try to take the rule space constraints into consideration until all flows are scheduled. We first convert all the non-zero x_s^f into 1 to indicate that one rule shall be embedded in $s \in S$ for flow $f \in F$ (line 16). After that, we initialize the unscheduled flow set $newFlowSet$ as empty set and start to check the flows that incur rule space constraints in lines 18–25. For each switch $s \in S$, we check whether converting all the non-zero x_s^s into 1 violates the rule space constraint or not. If there is enough space, we set the residual rule space as the space of the switch for next-round optimization (line 20); otherwise, we remove the flows with the least values of x_s^f and put f into $newFlowSet$ until s have enough space to accommodate the rest flows (lines 22–23). Accordingly, as s does not have enough space to accommodate more flows any more, we shall remove it from S for the next-round optimization. After we update the rule space for all switches and obtain the unscheduled flow set, we can re-solve **Com-Min-LP** with the updated system parameters to obtain new flow schedule solutions. Such routine proceeds until all flows get completed scheduled.

6 Performance Evaluation

In this section, we report our simulation based performance evaluation for our proposed relaxation heuristic algorithm ("Relax"), which is compared against the optimal solution ("Opt") and a greedy algorithm ("Greedy"). Commercial solver Gurobi[1] is used to solve our **Com-Min** and **Com-Min-LP** problems. In default, we consider a substrate network with $|N| = 5$ servers interconnected by

[1] http://www.gurobi.com.

$|S| = 3$ switches, where each switch is with $C_s = 5, \forall s \in S$. We then investigate how our algorithm performs under different scenarios with different settings and also how various system parameters affect the overall communication cost.

Algorithm 1. Relaxation-based Algorithm

```
1: Relax the binary variables $x_v$, solve the **Com-Min-LP** problem
2: Sort $x_v, \forall v \in V^e$ decreasingly into $\Pi_v$
3: for all Process $p \in P$ do
4:   while True do
5:     Find the first $x_v \in \Pi_v$ with $t(v) = p$
6:     Set the found out $x_v$ as 1 and the others with $t(v) = p$ as 0
7:     if The host server $h(v)$ has enough resource to accommodate process $t(v)$
       then
8:       break
9:     else
10:      Set the found out $x_v$ as 0 and remove it from $\Pi_v$
11:    end if
12:  end while
13: end for
14: Take the values of $x_v, \forall v \in V^e$ as known into
```

$$\textbf{Com-Min-LP}: \quad \min: \sum_{f \in F} \sum_{e_{uv} \in E^e} \lambda_{uv}^{f_{t(u)t(v)}} \cdot P_{n(u)n(v)},$$
$$\text{s.t.}: (1) - (9)$$
$$0 \le x_s^f \le 1, \forall f \in F, s \in S.$$

```
    and solve it to obtain the flow scheduling decisions
15: while Not all flows have been scheduled do
16:   Convert all the non-zero $x_s^f, \forall s \in S, f \in F$ to 1
17:   $newFlowSet \leftarrow \emptyset$
18:   for all $s \in S$ do
19:     if The switch $s \in S$ has enough rule space to accommodate the rules then
20:       Set the residual rule space as the rule space capacity of $s$
21:     else
22:       Remove the flows with the least values of $x_s^f$ until $s$ can accommodate all
          the flows, and put $f$ into $newFlowSet$
23:       Remove $s$ from $S$
24:     end if
25:   end for
26:   Resolve the **Com-Min-LP** with the updated rule space capacity constraints
      and updated flow set $newFlowSet$
27: end while
```

6.1 On the Effect of Rule Space

As our work intends to investigate how the rule space affects the dataflow computing program deployment and the corresponding communication cost, we first

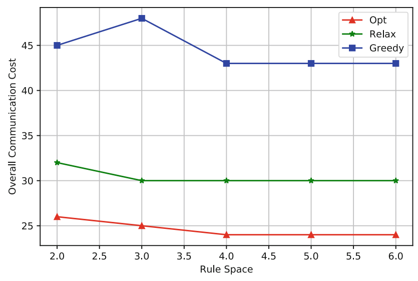

Fig. 2. The overall communication cost under different rule spaces on the SDN switches

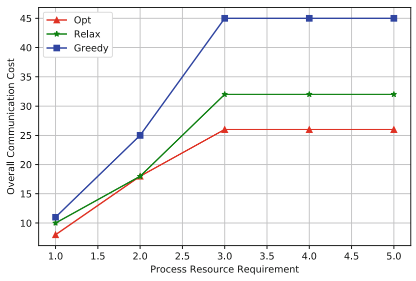

Fig. 3. The overall communication cost under different process resource requirements

conduct a series of experiments by varying the number of rule space in each switches from 2−6 and plot the experiments in Fig. 2. We can first notice that the overall communication cost achieved by our algorithm is much close the optimal solution, under any values of rule space. This validates the correctness and efficiency of our algorithm. We also further notice that the communication cost shows as a decreasing function of the rule space. This is because with more rule space, more flows can go through the switches with lower unit communication cost and hence the communication cost decreases. However, when the rule space is large enough, most flows already can go though the switches with lower unit communication cost, the overall communication cost becomes convergent. Such phenomenon proves that the rule space indeed has deep influence on the dataflow computing program deployment and hence the communication cost, especially when the rule space is severely limited.

6.2 On the Effect of Process Resource Requirement

Next, we check how our algorithm performs under different process resource requirements by varying the resource requirement of each process. The experiment results are reported in Fig. 3. Once again, we notice that our algorithm performs much close to the optimal solution. This verifies that our algorithm can well adapt to the changes of process resource requirement and always obtain sub-optimal solution. We also notice that the overall communication cost also increases with the process resource requirement. This is because, when the process resource requirement increases, the number of processes that can be hosted by one server decreases. This makes more processes go through the switches, inevitably incurring comparatively higher communication cost. Considering an extreme case when all processes can be accommodated in one server. In this case, no communication cost is incurred as all communications can go through intra-server communication.

7 Conclusion and Discussion

In this paper, we investigate how to deploy a dataflow computing program shared by multiple different flows in a distributed computing environment with communication unit cost diversity. Specially, we consider that the substrate servers are interconnected by SDN switches, whose forwarding capabilities are constrained by the limited rule space. With respect to such fact, we formulate the cost minimization problem into an MILP form and then accordingly propose a relaxation based heuristic algorithm in polynomial time. Through extensive simulation-based studies, the high efficiency of our heuristic algorithm is verified by the fact that it performs much close to the optimal solution.

Acknowledgment. This work was partially sponsored by the National Basic Research 973 Program of China under grant 2015CB352403, the National Natural Science Foundation of China (NSFC) (61602301, 61402425).

References

1. Abadi, M., et al.: TensorFlow: a system for large-scale machine learning. In: Proceedings of OSDI, vol. 16, pp. 265–283 (2016)
2. Botero, J.F., Hesselbach, X., Duelli, M., Schlosser, D., Fischer, A., De Meer, H.: Energy efficient virtual network embedding. IEEE Commun. Lett. **16**(5), 756–759 (2012)
3. Chowdhury, N.M.K., Rahman, M.R., Boutaba, R.: Virtual network embedding with coordinated node and link mapping. In: INFOCOM 2009, pp. 783–791. IEEE (2009)
4. Fischer, A., Botero, J.F., Beck, M.T., de Meer, H., Hesselbach, X.: Virtual network embedding: a survey. IEEE Commun. Surv. Tutor. **15**(4), 1888–1906 (2013)
5. Giroire, F., Moulierac, J., Phan, T.K.: Optimizing rule placement in software-defined networks for energy-aware routing. In: 2014 IEEE Global Communications Conference (GLOBECOM), pp. 2523–2529. IEEE (2014)
6. Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I.: Graphx: graph processing in a distributed dataflow framework. In: Proceedings of OSDI, pp. 599–613. USENIX, Broomfield (2014)
7. Gu, L., Zeng, D., Guo, S., Xiang, Y., Hu, J.: A general communication cost optimization framework for big data stream processing in geo-distributed data centers. IEEE Trans. Comput. **65**(1), 19–29 (2016)
8. Kang, N., Liu, Z., Rexford, J., Walker, D.: Optimizing the one big switch abstraction in software-defined networks. In: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, pp. 13–24. ACM (2013)
9. Kreutz, D., Ramos, F.M., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: a comprehensive survey. Proc. IEEE **103**(1), 14–76 (2015)
10. Murray, D.G., Schwarzkopf, M., Smowton, C., Smith, S., Madhavapeddy, A., Hand, S.: CIEL: a universal execution engine for distributed data-flow computing. In: Proceedings of ACM/USENIX NSDI, pp. 113–126 (2011)
11. Sun, G., Yu, H., Anand, V., Li, L.: A cost efficient framework and algorithm for embedding dynamic virtual network requests. Futur. Gener. Comput. Syst. **29**(5), 1265–1277 (2013)