Complexity analysis for DROPLET: Distributed Operator Placement for IoT Applications Spanning Edge and Cloud Resources

Sandur, Atul; Elgamal, Tarek; Agha, Gul; Nahrstedt, Klara

1 Introduction

The continued growth of Internet-of-Things (IoT) applications requires the ability to extract insights from massive amounts of data streams observed in real-time. Such data is collected from multiple input sources including surveillance cameras, wearable devices, etc. Due to limited computational capability of devices collecting such data, the conventional approach of performing analytics is to send data streams to a cloud platform and perform analytics tasks remotely. However, given the tremendous amount of data transfer required by video data as an example, the latency and bandwidth requirements become significantly high.

A novel approach being explored is to leverage *edge devices* that act as gateways to aggregate and forward IoT-captures data. They are typically few hops away from the data source, with non-trivial computational and network resources.

IoT applications consist of set of dependent operations modeled as dataflow graphs. Each operation describes some computation on the incoming data such as convolution, encryption or filtering. So given the edge devices, a key challenge is to automatically decide how to partition such operations among edge and cloud compute resources, in order to minimize the overall completion time of the entire graph of operations. We refer to this problem as *distributed operator placement*.

Note that the data for IoT applications comes from various sources as continuous streams of observations, so a queuing delay exists for observations within the same data stream and across streams sharing compute/network resources. Droplet [ET18] optimizes execution time of dataflow operations without violating dependency constraints by: (1) considering computation, communication and queuing delays, capturing pipeline parallelism in the execution model and, (2) proposing a scalable dynamic programming algorithm to solve operator placement in log-linear time with respect to number of operators.

In this technical report, we show that the distributed operator placement problem as defined in [ET18] is NP-complete. [ET18] proposes a dynamic programming based heuristic solution Droplet for the placement problem. We provide detailed derivation for complexity of Droplet algorithm in this report. The rest of the report is structured as follows: section 2 defines the system models and notation used in the rest of the report. Section 3 defines the distributed operator placement problem. Section 4 proves that our placement problem is NP-complete and finally section 5 provides derivation of complexity of Droplet. The report ends with a conclusion in 6

2 Models and Problem Definition

Resource Model: We model the physical resources on which we execute operators as a weighted directed graph $G_R = (V_R, E_R)$, as shown in Figure 2 where the vertices $V_R = \{E, C\}$ represent two compute resources one edge E and one cloud C. Note that in this report, we will consider systems of two resources edge and cloud. The links $E_R = \{(B_{E,C}, B_{C,E})\}$ represent bandwidths from edge to cloud and from cloud to edge. An example resource graph is given in Figure 2. Each physical resource can have multiple virtual resources that can use a portion of the physical resources' CPU and memory. Virtual resources can be thought of as virtual machines (VMs), or containers. For simplicity, we refer to virtual resources as containers in this paper. Each of E and C has maximum number of containers denoted as m_E, m_C respectively. We assume that all containers in the same physical resource are homogeneous in terms of compute and memory resources reserved for them.

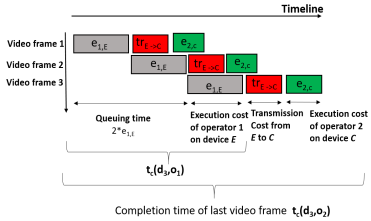

Figure 1: Pipelined Execution of operator o_1 on resource E and o_2 on resource C.

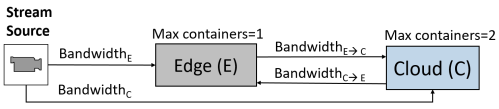

Figure 2: Example resource graph.

IoT Data Model: We model the IoT data as an unbounded stream S of *data frames*, $s = \langle d_1, d_2, \cdots \rangle$. A *data frame* can be seen as one unit of data or a measurement defined by the application (e.g. tuple, video frame, sensor reading). The IoT application is typically defined for a stream/sequence of such data frames with fixed frequency f (e.g., 30 frames/s). We aggregate a sequence of data frames into *chunks* where each chunk has duration T. The k^{th} chunk ch_k can be defined as $ch_k = \langle d_1^k, d_2^k, \ldots d_I^k \rangle$, where $I = f \cdot T$ is the chunk size. In the rest of the paper we deal with chunks of data. The chunk duration/size is an application defined parameter.
Operator Graph Model: We model the dataflow in an IoT application as a directed-acyclic-graph (DAG) $G_o = (V_o, E_o)$ of operators (Fig. 3). An operator is a processing element that can execute user-defined code (e.g. convolution or face detection in face recognition applications). The vertices in the DAG represent operators $V_o = \{o_i \mid i = 1 \ldots n\}$, and the links between them $E_o = \{o_i \rightarrow o_j \mid i \neq j \ \wedge \ 1 \leq i, j \leq n\}$ represent the dataflow dependencies, where $o_i \rightarrow o_j$ means o_i is applied to a data frame before o_j is applied to the same frame.
Operator Profile: Each operator in Figure 3 is associated with a *profile* which includes:

1. The cost of executing operator o_i on a data frame d_j, when placed on each resource $r_l \in V_R$. Operator o_i can be placed on resource E or C so we denote their corresponding execution costs as $e_{i,E}$ and $e_{i,C}$.

   We assume that the data frames within chunk ch_k have equal sizes therefore the execution cost is the same for all data frames.

2. The size of output data of operator o_i and we denote it as D_{o_i} bytes. The transmission time of the result of o_i from the edge to the cloud, $tr(E \xrightarrow{D_{o_i}} C) = \frac{D_{o_i}}{B_{E,C}}$, where $B_{E,C}$ is the bandwidth (bytes/sec) from the edge to the cloud. Since we assume that the data frames within a chunk have equal sizes, the output data of o_i is the same for all data frames i.e. $\forall d_j^k \in ch_k \ D_{o_i} = D_{o_i}(d_j^k)$.

Execution model: One of the key contributions of [ET18] is the ability to model concurrent execution of multiple dependent operators in a pipelining fashion. Consider the example in Fig. 1. A chunk of 3 video frames is processed by two operators o_1 and o_2. o_1 is placed on resource E and o_2 on resource C. First o_1 is applied to the first frame and the output is transmitted to resource C where operator o_2 can be applied, while o_1 is concurrently applied to the second frame. Assuming $e_{1,E}$ is a large value, the time for processing the entire chunk ($T(ch_k)$) is equivalent to $t(d_j^k, o_i)$, which is the completion time of the last data frame d_j^k in ch_k and o_i is the last operator applied to it. So for the above example, $T(ch_k)$ is:

$$t(d_3, o_2) = \underbrace{t(d_3, o_1)}_{\text{prev. operator}} + \underbrace{0}_{\text{queuing}} + \underbrace{tr(E \xrightarrow{D_{o_1}} C)}_{\text{transmission}} + \underbrace{e_{2,C}}_{\text{execution}}$$

$$t_c(d_3, o_1) = \underbrace{0}_{\text{prev. operator}} + \underbrace{2 \cdot e_{1,E}}_{\text{queuing}} + \underbrace{0}_{\text{transmission}} + \underbrace{e_{1,E}}_{\text{execution}}$$

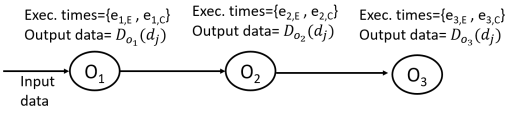

Figure 3: Example operator DAG.

3 Problem Definition:

Let M be a map from operators to resources defined as $M : V_o \rightarrow V_R, M(o_i) = r_j \mid r_j \in \{E, C\}$. And let $T(ch_k, M)$ be the completion time of processing an entire chunk ch_k given the mapping M. The operator placement problem is to find a map M for which $T(ch_k, M)$ is minimized subject to the following constraints:

- If o_i and $o_{i'}$ are dependent operators then:

$$(M(o_i) = r_l) \ \wedge \ (M(o_{i'}) = r_p), \implies B(r_l, r_p) \neq 0 \tag{1}$$

- Total resource cost of containers consumed by operators executing on resource r_j does not exceed $Capacity(r_l)$, which defines the available capacity of resource r_l

$$\forall l \mid r_l \in E, C, \sum_{i=1}^{i=n} \mathbb{1}_M(o_i, r_l) \cdot \alpha_l \leq Capacity(r_l) \tag{2}$$

  Here α_l is the penalty for consuming a container in resource r_l. Note that each physical resource would have a limit on the total number of containers that it can host.

The chunk completion time $T(ch_k, M)$ can be defined in terms of the completion time of last data frame d_I in ch_k when processed by the final operator o_n as:

$$\begin{aligned} T(ch_k, M) = t(d_I, o_n) = \underbrace{t(d_I, o_{n-1})}_{\text{prev. operator completion}} + \underbrace{t_{queue}(d_I, o_n)}_{\text{queuing time}} + \\ + \mathbb{1}_M(o_{n-1}, r_k) \cdot \underbrace{\mathbb{1}_M(o_n, r_l) \cdot tr(k \xrightarrow{D_{o_{n-1}}} l)}_{\text{transmission cost}} + \underbrace{\mathbb{1}_M(o_n, r_l) \cdot e_{n,l}}_{\text{execution cost}} \end{aligned} \tag{3}$$

where

$$\mathbb{1}_M(o_i, r_l) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } o_i \text{ is placed on resource } r_l \\ 0 & \text{otherwise} \end{cases}$$

$$t_{queue}(d_j, o_i) \stackrel{\text{def}}{=} \text{queuing time of frame } d_j \text{ on operator } o_i$$

Here, $t_c(d_I, o_0) = 0$ is the base case for recursive equation 3 and it is the time just before first operator o_1 starts.

The major challenge to solve the operator placement problem is that $T(ch_k, M)$ (Eq. 3) depends on the mapping M of all the operators that are running concurrently in a pipelined fashion. Hence, an exhaustive search of the solution space is required to determine the optimal values of M (in $O(2^n)$ time). There have been approaches proposed, such as branch-and-bound [TLG16], to decrease the search complexity. But in the worst case, it is still exponential in n.

4 NP-completeness

Except for 3 restrictive cases which involve constraints on the resources/DAG structure, DAG scheduling in general is NP-complete [K+99]. Our problem does not meet any of the 3 conditions. We now transform a previous DAG scheduling problem that is proved to be NP-complete into ours in polynomial time. Let $\mathcal{T} = \{T_1, T_2, \cdots T_n\}$ be a set of tasks that have partial order to form a DAG, and $\mathcal{R} = \{R_1, R_2, \cdots R_n\}$ be resources with each having bounded capacity $\mathcal{B}_\backslash$. Each task T_i has a latency time τ_i and a capacity requirement of $R_j(T_i) \leq B_j$ when run on resource R_j. [G+89]

has earlier shown that checking the existence of a valid schedule from tasks $\mathcal{T}$ to resources $\mathcal{R}$ while meeting a deadline D, the resource capacity bounds and partial task ordering is NP-complete for more than two resources [GS16]. We can get the fastest schedule by testing different integer values of D to find the smallest with a valid schedule.

We transform this known NP-complete problem to our optimization problem, which is more complex, in polynomial time, as follows. We map each task $T_i \in \mathcal{T}$ to an operator $o_i \in V_o$, with running time τ_i replacing the computation, communication and queuing cost of o_i. We also transform the resource bounds B_j into the compute bounds in constraint 2 above. Given the mapping M, we can test its validity as a solution by checking constraints 1 and 2 at each operator and finding critical path in the DAG in $\mathcal{O}(\mid E_o \mid + \mid V_o \mid + (\mid V_R \mid \times \mid V_o \mid) + (\mid V_o \mid + \mid E_o \mid))$, which is $\mathcal{O}((\mid V_R \mid \times \mid V_o \mid) + \mid E_o \mid)$. This is polynomial and hence shows that our optimization problem is NP-complete.

5 Complexity analysis of Droplet

Given a dataflow graph $G_o = (V_o, E_o)$ and resource graph G_R as input, in order to determine Droplet's complexity, we need to find the cost of (1) topological sort, (2) generating the vertices and edges in our placement graph, (3) computing edge weights and (4) running shortest path algorithm for the constructed placement graph.

For the graph G_o, topological sort can be done in $\mathcal{O}(\mid V_o \mid + \mid E_o \mid)$. So we can get a list of operators L_o from this first step in our algorithm. Let $f_s(L_o, k)$ be a function which computes the count of distinct sublists of size k, from list L_o. It follows that $f_s(L_o, k)$ can be written as:

$$f_s(L_o, k) = \mid L_o \mid -k + 1 \tag{4}$$

where $\mid L_o \mid$ denotes the length of list L_o. Also note that

$$\mid L_o \mid = \mid V_o \mid \tag{5}$$

Now, given L is the number of resources available for placement, the number of vertices n_v that can be generated in our placement graph from the set L_o is (from equations 4 and 5):

$$n_v = \sum_{k=1}^{L} (f_s(L_o, k) \cdot {}^{L}P_k) = \sum_{k=1}^{L} \mid V_o \mid \cdot {}^{L}P_k - \sum_{k=1}^{L} k \cdot {}^{L}P_k + \sum_{k=1}^{L} {}^{L}P_k \tag{6}$$

Note that ${}^{L}P_k$ represents permutations for the number of ways to obtain an ordered subset of k elements, from a set of L elements.

From [SP], we know the summation on permutations in first and last terms of equation 6 evaluates to:

$$\sum_{k=1}^{L} {}^{L}P_k \approx L! \cdot e \tag{7}$$

We can also show that the second term in equation 6 evaluates to:

$$\begin{aligned}
\sum_{k=1}^{L} k \cdot {}^{L}P_k &= L! \cdot [\frac{L}{0!} + \frac{L-1}{1!} + \cdots + \frac{1}{(L-1)!}] \\
&= L! \cdot [L + \frac{L}{1!} + \cdots + \frac{L}{(L-1)!} - [\frac{1}{1!} + \frac{2}{2!} + \cdots + \frac{L-1}{(L-1)!}]] \\
&= (L-1)[L! \cdot e - \frac{L!}{(L-1)!}] + L^2 \\
&\approx (L-1)[L! \cdot e] + L \qquad \text{- - - from equation 7}
\end{aligned} \tag{8}$$

From equations 6, 7 and 8, we are able to show that the number of vertices n_v can be written as:

$$n_v \approx \mathcal{O}(\mid V_o \mid \cdot L!) \tag{9}$$

We now need to determine the number of edges in our placement graph. Based on Droplet algorithm, a constraint on the first operator of a child node has to be satisfied, for it to share

an edge with it's candidate parent- the resource on which first operator of a child node is placed, should appear in the candidate parent node. Based on this constraint, the number of such child nodes for a vertex with k operators is determined to be:

$$n_{childnodes} = \sum_{p=1}^{|V_o|-k} k \cdot {}^{L-1}P_{p-1} \tag{10}$$

Equation 10 gives the number of edges for each vertex with k operators in our placement graph. Combining equations 6 and 10 and and assuming the number of operators is greater than the number of resources i.e. $\mid V_o \mid \gg L$, the total number of edges is:

$$\begin{aligned} n_{edges} &= \sum_{k=1}^{L}(f_s(L_o,k) \cdot {}^{L}P_k \cdot (\sum_{p=1}^{|V_o|-k} k \cdot {}^{L-1}P_{p-1})) \\ &\approx \sum_{k=1}^{L}(f_s(L_o,k) \cdot {}^{L}P_k \cdot k \cdot (L-1)! \cdot e) \qquad \text{- - - from equation 7} \\ &= (L-1)! \cdot e[\mid V_o \mid ((L-1)(L! \cdot e)+L) - \sum_{k=1}^{L} k^2 \cdot {}^{L}P_k + (L-1)(L! \cdot e) + L] \end{aligned} \tag{11}$$

The last step comes from substituting the summation term from equation 8. Using equation 7 and expanding square terms below, we now observe the following result to expand equation 11:

$$\begin{aligned} \sum_{k=1}^{L} k^2 \cdot {}^{L}P_k &= L! \cdot [\frac{1}{(L-1)!} + \frac{2^2}{(L-2)!} + \cdots + \frac{L^2}{0!}] \\ &= L![L^2[1 + \frac{1}{1!} + \frac{1}{2!} + \cdots \frac{1}{(L-1)!}] - 2 \cdot L \cdot [\frac{1}{1!} + \frac{2}{2!} + \cdots \frac{n-1}{(n-1)!}] + \\ &[\frac{1^2}{1!} + \frac{2^2}{2!} + \cdots \frac{(L-1)^2}{(L-1)!}]] \\ &\approx L^2 \cdot (L! \cdot e) - 2 \cdot L \cdot L![e - \frac{1}{(L-1)!}] + [1 + \frac{2}{1!} + \cdots + \frac{L-1}{(L-2)!}]L! \\ &= L^2 \cdot (L! \cdot e) - 2 \cdot L \cdot L! \cdot [e - \frac{1}{(L-1)!}] + 2 \cdot L! \cdot e - L^2 - 3 \cdot L \\ &= ((L-1)^2 + 1) \cdot L! \cdot e + L^2 - 3 \cdot L \end{aligned} \tag{12}$$

The fourth step in equation 12 is obtained by simply expanding the factorial terms in previous step. Using equation 12, we can expand the terms in equation 11 to be:

$$\begin{aligned} n_{edges} &= (L-1)! \cdot e[(\mid V_o \mid +1)((L-1) \cdot (L! \cdot e) + L) - (((L-1)^2 + 1)(L! \cdot e) + L^2 - 3 \cdot L))] \\ &= (L-1)! \cdot e[\mid V_o \mid \cdot L(L! \cdot e) - \mid V_o \mid \cdot (L! \cdot e) + L \cdot \mid V_o \mid + L - L^2 \cdot (L! \cdot e) + 3L \cdot (L! \cdot e) \\ &- 3(L! \cdot e) - L^2 + 3 \cdot L] \end{aligned} \tag{13}$$

Looking at the dominant terms in the above equation for n_{edges}, it can be computed in $\approx \mathcal{O}(\mid V_o \mid \cdot (L!)^2)$.

For each edge, we need to compute it's edge weight. If an edge has source vertex with k operators, the execution time of each operator combined with transmission and queuing delays between operators results in k operations that need to be performed for calculating it's edge weight. So the cost of computing all edge weights from equation 11 would be:

$$c_{edgeweight} = \sum_{k=1}^{L}(f_s(L_o,k) \cdot {}^{L}P_k \cdot k^2 \cdot (L-1)! \cdot e) \tag{14}$$

We can expand terms in equation 14 similar to how we did it for equation 11 to show that it can be done in $\mathcal{O}(\mid V_o \mid \cdot (L!)^2 \cdot L)$

Following are costs for individual steps in Droplet algorithm: (1) topological sort ($\mathcal{O}(\mid V_o \mid + \mid E_o \mid)$ where $\mid V_o \mid$ and $\mid E_o \mid$ are the number of operators and links in operator graph respectively), (2) generating vertices ($\mathcal{O}(\mid V_o \mid \cdot L!)$), (3) generating edges ($\mathcal{O}(\mid V_o \mid \cdot (L!)^2)$), (4) computing edge weights ($\mathcal{O}(\mid V_o \mid \cdot (L!)^2 \cdot L)$), (5) running shortest path algorithm [F+87] on constructed placement graph ($\mathcal{O}(\mid V_o \mid \cdot L! \cdot log(\mid V_o \mid \cdot L!) + \mid V_o \mid \cdot (L!)^2)$). The overall complexity of Droplet can thus be given as:

$$\mathcal{O}(\mid V_o \mid \cdot (L!) \cdot (log(\mid V_o \mid \cdot L!) + L! \cdot L) + \mid E_o \mid) \tag{15}$$

6 Conclusion

In this report, we showed that the distributed operator placement problem described in [ET18] is NP-complete. This motivates the need for a heuristic approach to solve the problem, which is done using dynamic programming techniques in [ET18]. In the report, we further show the dynamic programming solution Droplet to be log-linear in the number of operators in our input operator graph. Experimental results in [ET18] further confirm this analysis. We thus analyze a heuristic solution to the distributed operator placement problem which is scalable in the number of operators and works well in practice.

References

[ET18] Agha G. Nguyen P. Nahrstedt K. Elgamal T., Sandur A. Droplet: Distributed operator placement for iot applications spanning edge and cloud resources. *IEEE Cloud 2018*, 2018.

[F+87] Michael Fredman et al. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, July 1987.

[G+89] M. R. Garey et al. Complexity results for multiprocessor scheduling under resource constraints. IEEE Computer Society Press, 1989.

[GS16] Rajrup Ghosh and Yogesh Simmhan. Distributed scheduling of event analytics across edge and cloud. *CoRR*, abs/1608.01537, 2016.

[K+99] Yu-Kwong Kwok et al. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4), 1999.

[SP] N. J. A. Sloane and S. Plouffe. The encyclopedia of integer sequences. *Academic Press*, abs/1608.01537.

[TLG16] L. Tong, Y. Li, and W. Gao. A hierarchical edge cloud architecture for mobile computing. In *IEEE INFOCOM 2016*, 2016.