An Operator Placement Algorithm for Complex In-Network Processing

Andreas Lachenmann, Venkatesh Bommasandra Sadasiva, and Frank Siegemund

European Microsoft Innovation Center GmbH, Germany
andreasl@microsoft.com, bs.venkatesh@gmail.com, franksie@microsoft.com

***Abstract*—In sensor network applications that do complex in-network processing, each node may perform its own task rather than all nodes executing the same aggregation function. To support the development of such applications, this paper introduces a network-level programming model that is based on the data flow in the application. When deploying the application, such a high-level model has to be mapped to nodes in the specific network topology. This paper presents an operator placement algorithm that tries to find an optimal mapping that minimizes data transmission throughout the network.**

Keywords—wireless sensor network; operator placement; data flow

I. INTRODUCTION

If sensor network applications do more complex in-network processing than simple aggregation, it is unlikely that all nodes will perform the same task. For example, in an elderly care scenario [1], there are specialized nodes that detect the activity of a patient using vibration sensors in the floor to analyze the patient's movements and sensors in furniture to determine when a cabinet has been opened.

If not all nodes perform the same tasks, programming the network with a database abstraction [2] does not seem to be practical since the individual processing tasks cannot be specified in a single SQL-like query. Furthermore, although it is possible to write custom implementations of the application on a node-per-node basis using the programming interface of a sensor network operating system, creating a separate code image for each node and deploying it in the network is a tedious task. Therefore, there is a need for a more high-level programming model that addresses these problems.

Our proposed programming model is based on data flows: In the application, data from the sensor nodes is sent through processing operators towards a sink. For instance, in the elderly care scenario, individual nodes have light sensors to detect if the fridge and cabinets in the kitchen are opened or a pressure sensor that checks if a chair is occupied. As a simplified example, the application can infer if the patient has prepared a meal and eats it by combining the sensor values (i.e., the fridge and cabinets have been opened and the chair is occupied some time after opening the supply cabinets).

Similar to other macroprogramming approaches [3], the developer defines the functionality of the complete network rather than the functionality of individual nodes. However, the efficiency of such an application largely depends on how it is mapped to the actual network topology. Therefore, in this paper, we present an algorithm that optimizes the placement of operators for a given network topology. Our algorithm builds on an existing algorithm for task scheduling from distributed systems [4] and transfers it to the area of sensor networks. Unlike existing approaches [5], our solution finds an optimum solution in polynomial time if the data flow graph is a tree, deals with lossy radio links, and approximates a solution for data flows that have a general graph structure.

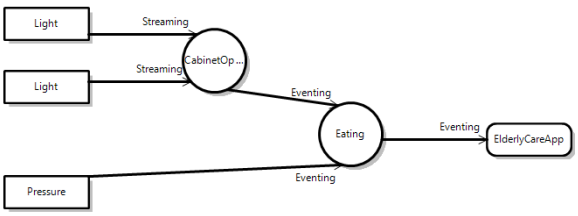

Fig. 1. Example of a data flow graph

The rest of this paper is structured as follows. In Section II we introduce our programming model and in Section III we describe our placement algorithm. In Section IV we present evaluation results. Finally, Section V concludes this paper.

II. PROGRAMMING MODEL

In our programming model, a sensor network application consists of different operators that are connected to form a data flow graph. Rather than developing the application for an individual node, this data flow graph refers to the network-wide data flow in the whole application. It starts at data sources (i.e., sensors) and specifies how this data is transformed in operators and consumed by data sinks. As an example, Fig. 1 shows the data flow graph of the (simplified) elderly care application introduced in Section I in our graphical modeling tool. It contains three different sensors whose output is processed by two operators. The results of the "Eating" operator flow into the sink application.

The data flow graph described by a programmer is independent of a concrete network topology. Instead, it is part of the deployment process to map the data flow graph onto concrete nodes. These nodes are organized in a physical topology graph that is used by the deployment tool. Nevertheless, the developers can still influence the placement of operators by defining additional information in the form of metadata.

We have implemented our programming model for the

This work is partially financed by the European Commission under the Framework 6 IST Project "Wirelessly Accessible Sensor Populations (WASP)".

Gather information about network topology
Remove cycles from undirected data-flow graph
    Find cycle
    Remove edge on cycles with least weight
    Restore tree properties
Build assignment graph
    Create node for each sensor node each operator can be placed on
    Connect nodes according to topology graph
    Compute weights of edges, considering data-loss rates
Find minimal-cost placement
    Repeat
        For each terminal node
            Compute shortest paths to all nodes in first fork set reached
        For each fork set that is reachable from all terminal nodes below it
            Store the shortest paths to all those terminal nodes
            Replace everything below with a new terminal node
            Set the edge weights to this new node to the sum of the shortest paths to the corresponding node in the fork set
    Until root reachable from all terminal nodes without fork set in between
Place operators on nodes on the selected shortest paths

Fig. 2. Overview of the algorithm

IMote2 sensor nodes running the .NET Micro Framework. Our implementation is integrated into Microsoft Visual Studio such that the developer can use a widespread IDE.

III. Operator Placement Algorithm

Fig. 2 gives a brief overview of our operator placement algorithm as it is described in this section.

A. Network Topology

Our placement algorithm assumes to have a global view of the network topology. This information is readily available in many applications. For example, if the nodes do not move, they can measure the loss rates to their neighbors over long periods of time. Each node just forwards the information about its direct neighbors to the sink node, which then computes the overall network topology. This is possible with reasonable overhead for small to midscale networks.

To verify this assumption, we simulated random topologies of up to 80 nodes and counted the amount of data transmitted. In these simulations, all nodes combined have to send just a few thousand bytes. Furthermore, using data about energy consumption of the commonly used CC2420 radio chip, each node in the experiments spends less than 5 mJ for sending the topology information. Therefore, the energy consumption should be negligible compared to battery capacities of several thousand Joules [6]. Using a centralized approach does not add a new single point of failure because in the applications we consider the sink would already be such a single point of failure [7] independent of the operator placement algorithm.

B. Data Flow Trees

The algorithm we build on [4] was originally developed to find an assignment of tasks to nodes in a distributed system such that the execution and communication costs are minimized. Transferred to our placement scenario, it requires that the data flow graph must be a tree with the data sink as the root. For trees, the algorithm guarantees to find an optimal solution in $O(mn^2)$ time for m operators and n sensor nodes.

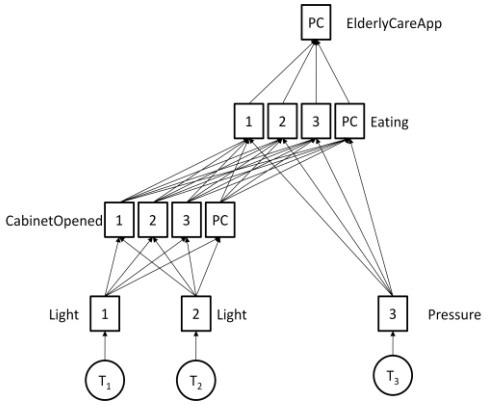

Fig. 3. Assignment graph for the example application

The algorithm creates a so-called assignment graph based on the information from the data flow graph and the topology graph. For example, for the simple data flow graph shown in Fig. 1 and a topology with three nodes and one PC, the assignment graph in Fig. 3 is created. For each combination of nodes of the data flow graph and the nodes in the topology graph it can be placed on, this graph contains a separate node.

The edges that connect the nodes in this graph are weighted with the costs for communication between the sensor nodes (see below). Furthermore, in the assignment graph so-called terminal nodes are added as leaves. The algorithm looks for the shortest paths from the terminal nodes to each node in a so-called fork set. A fork set corresponds to a node in the data flow graph with an incoming degree that is greater than one. The algorithm temporarily removes the nodes connected to the fork set. However, it keeps the information about the shortest paths from all terminal nodes. Then it adds an artificial terminal node that is connected to the fork set. The weights of the edges connecting the fork set with this node correspond to the sum of edge weights for the shortest paths.

For example, in Fig. 3 the "CabinetOpened" operator is a fork set. After computing the shortest paths from the terminal nodes T_1 and T_2 to each node in this fork set, the nodes below "CabinetOpened" are removed. A new terminal node T is introduced and connected to all nodes of the fork set. The weights of these edges correspond to the sum of the shortest paths' weights from both T_1 and T_2 to each node "CabinetOpened" can be placed on. Using this modified graph, the process is repeated until the root is reached. Then the nodes are mapped to all nodes on the shortest paths that have been previously removed from the graph.

As already shown in Fig. 3, some parts of the application such as the data sources can only be placed on specific nodes. For example, these might be the only nodes that have a given type of sensor. To deal with these situations, in the assignment graph, a node is only created for the sensor nodes onto which a part of the application can be deployed. Therefore, the algorithm is suitable for heterogeneous networks.

C. Edge Weights in the Assignment Graph

The weights of the edges in the assignment graph are used

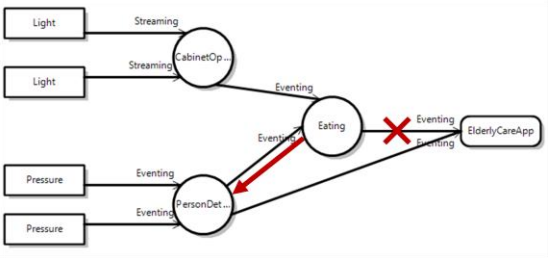

Fig. 4. Example for multiple outgoing edges

to express the costs for communicating with other nodes. The cost metric has to consider both the amount of data sent between two operators as well as the number of hops it has to be forwarded and the loss rates of the links. We assume that the developer has an estimate about how often an operator sends data and that the size of the output data type is known. Furthermore, we assume that links are made reliable with link layer acknowledgments and retransmissions.

Similar to the ETX routing metric [8], the expected number of (re-)transmissions $t_{a,b}$ between node a and b can be computed as $\frac{1}{1-l_{a,b}}$, where $l_{a,b}$ refers to the loss rate which is delivered as a part of the topology information described in Subsection A. The expected number of transmissions $T_{a,z}$ along the shortest routing path that connects nodes a and z is $t_{a,b} + t_{b,c} + \cdots + t_{y,z}$. At operator i, the metric can be expressed as the following function $Cost_i(p)$ for a placement p:

$$Cost_i(p) := \sum_{j \in Inputs_i} T_{p(j),p(i)} \cdot s_j \cdot f_j$$

This function computes the cost for each operator and data source j that feed input into operator i. For each such operator, it takes the expected transmission costs $T_{p(j),p(i)}$ on the shortest path between the nodes assigned to the operators in the current placement p. It multiplies this number with the size s_j of the output data of operator j and the expected frequency f_j. By considering the expected number of retransmissions, the algorithm takes into account lossy links.

D. General Data Flow Graphs

The algorithm makes the following assumptions that cannot be made for general data flow graphs. It assumes that there are no cycles in such graphs and that each node has at most one outgoing edge. Both assumptions can be removed in the same way but, for efficiency, we cannot get the optimum solution.

In both cases we look for cycles in the undirected graph that consists of the same operators as the data flow graph and that has an edge between two operators if there is an edge between the corresponding ones in the data flow graph. If a cycle is found, we remove the edge on the cycle over which the least amount of data will be transmitted. This edge will probably have only a small effect on operator placement and can be neglected. Although the edge will not be considered for operator placement, it is, of course, still present in the application.

After removing an edge from the data flow graph, some nodes might no longer have a path to the sink. Therefore, the direction of some edges in the data flow graph has to be reversed. In particular, nodes that do not have a path to the sink have to reverse the direction of edges on the broken cycle such that no node has more than one outgoing edge and such that there is a path from each node to the sink. On this graph, our standard placement algorithm from Subsection B can then be executed. Since it minimizes the overall weight of the graph, the direction of the edges is not important for the algorithm.

This algorithm has to be repeated on the undirected graph until there is no longer a cycle in it. Fig. 4 illustrates the algorithm with an example that extends the simple elderly care application with another operator, which detects the presence of a person on a chair. The results of this operator can also be used to, e.g., determine if visitors are present. Therefore, in the data flow graph shown here, there are two outgoing edges from the "PersonDetection" operator. If all edges are considered to be undirected, there is a cycle between the "Eating" and "PersonDetection" operators and the sink. Our algorithm removes the edge with the least cost (this is presumed to be the edge between "Eating" and the sink). If the direction of the edge between "PersonDetection" and "Eating" is reversed, the result is a tree that can be processed by our algorithm.

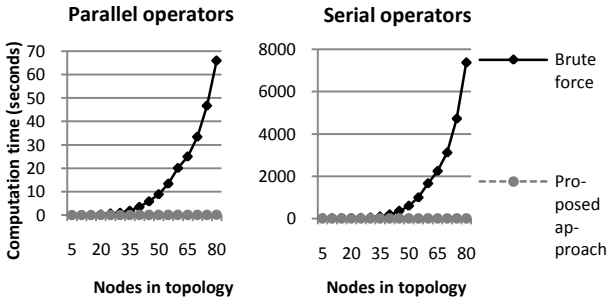

Fig. 5. Efficiency of the operator placement algorithm

IV. Evaluation

A. Efficiency of the Algorithm

To evaluate the efficiency of our algorithm, we run it on a PC as part of our Visual Studio extension. Fig. 5 compares the efficiency of our operator placement algorithm with a brute force approach that simply checks the costs of each placement possible. We have done these experiments for several network topologies with randomly placed nodes and two different data flow graphs. The first data flow graph consists of three operators, each with two inputs from data sources, that are all directly connected to the sink application. Therefore, in this graph the operators are connected in parallel. The second data flow graph is a serial variant where three operators are placed in a chain, having two inputs from data sources and one from the previous operator in the chain. In addition, to increase the complexity of the example, another operator with two sensor inputs feeds into the sink application.

As Fig. 5 shows, the computation time for the brute force algorithm increases exponentially. For the serial data flow graph, which has just four operators in total, it takes more than two hours to compute the optimal placement for a sensor network of 80 nodes; our proposed algorithm delivers its result

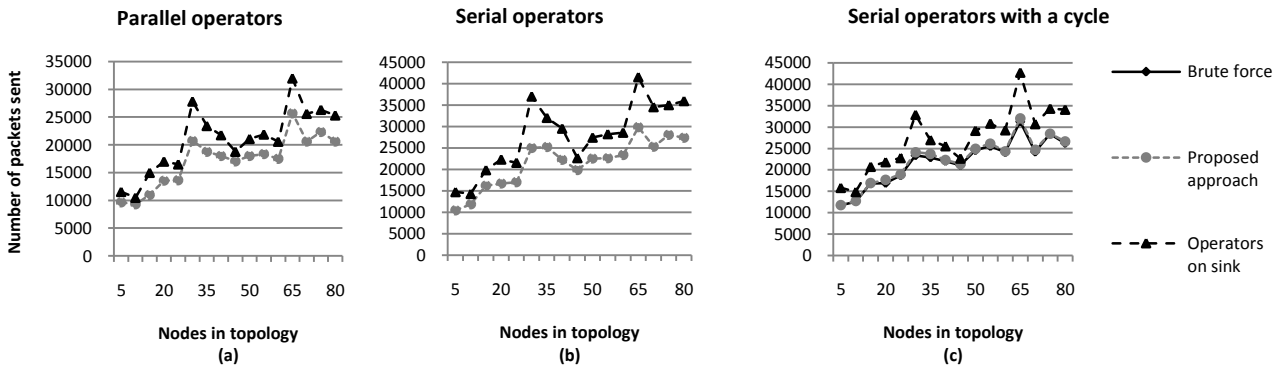

Fig. 6. Quality of operator placement in terms of number of packets sent in the whole network

after only 0.6 s. For the simpler parallel data flow graph all computations of our approach take less than 0.14 s.

B. Quality of Operator Placement

For evaluating the quality of the mappings, we simulate a sensor network application that runs for 10 days where each node sends 5 packets per hour towards the sink. Using a custom simulator, we sum up the total number of packets generated and forwarded by all nodes. To show the benefit of our algorithm, we compare the quality of its placement, i.e., the number of packets sent throughout the network with an approach where all operators are placed on the sink. Fig. 6 (a) and (b) show the average values of 50 runs when varying the number of nodes in the network topology. Here we use the same data flow graphs as in Subsection A.

On average, the placement on the sink leads to an overhead of 25 % compared to our optimal solution. Because the number of operators is relatively small in these examples and because the data sources and sinks are distributed randomly throughout the network, the benefit of in-network processing could be even greater for real-world deployments. Furthermore, in our simulations we assume only a small difference in the data size of incoming and outgoing data of an operator, which might be larger in a real-world application.

C. Quality for General Graphs

For general graphs, our algorithm can no longer provide the guarantee to find the optimal placement. Nevertheless, as Fig. 6 (c) shows for an exemplary data flow graph, the placement is still almost optimal when compared to a brute force approach. The data flow graph used here is similar to the serial data flow graph of Subsection A but includes a cycle between the chain of operators. Although our algorithm removes an edge to restore the tree properties, the results are almost identical to the brute force approach that does not apply this simplification. On average, the solution found by our proposed algorithm is just 1.7 % greater than the optimal placement.

V. Conclusions and Future Work

In this paper we have described a network-level programming model which is based on the data flow in the whole sensor network. When deploying applications in a sensor network, the data flow has to be mapped to the concrete topology. Such a mapping is crucial for efficient execution that minimizes the amount of data transmitted. As we have shown in the evaluation, using a naïve approach that checks out all possible solutions is not even practical for very small data flow graphs.

Although our operator placement algorithm is based on a well-known algorithm from distributed systems, some properties of sensor networks and of the envisioned applications required changes to that algorithm. For example, we added support for heterogeneous networks and for properties of the physical environment by limiting the nodes an operator can be placed on. In addition, with our cost metric we specifically support lossy links where packets might have to be retransmitted. Finally, we support data flow graphs that are not trees because with think that they will be found frequently in sensor network applications. In summary, our operator placement algorithm efficiently finds a solution for the operator placement problem that minimizes the amount of data transmitted in the network.

Regarding future work, we plan to make our algorithm adaptive so that it can react to changes in the topology without having to recompute the complete placement.

References

[1] L. Atallah, B. Lo, G.-Z. Yang and F. Siegemund, "Wirelessly Accessible Sensor Populations (WASP) for Elderly Care Monitoring," in *Proc. of the Worksh. on Ambient Techn. for Diag. and Monit. Chronic Patients,* 2008.

[2] S. R. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong, "TinyDB: An acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.,* 30, 2005, pp. 122-173.

[3] R. Sugihara and R. K. Gupta, "Programming models for sensor networks: A survey," *ACM Trans. on Sensor Netw.,* 4(2), 2008, pp. 1-29.

[4] S. H. Bokhari, "A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System," *IEEE Trans. on Softw. Eng.,* SE-7(6), 1981, pp. 583-589.

[5] B. J. Bonfils and P. Bonnet, "Adaptive and Decentralized Operator Placement for In-Network Query Processing," in *Proc. of the 2nd Intl. Worksh. on Inform. Proc. in Sensor Netw.,* 2003, pp. 47-62.

[6] A. Lachenmann, P. J. Marrón, D. Minder and K. Rothermel, "Meeting Lifetime Goals with Energy Levels," in *Proc. of the 5th ACM Conference on Embedded Networked Sensor Systems,* 2007, pp. 131-144.

[7] B. Raman and K. Chebrolu, "Censor Networks: A Critique of "Sensor Networks" from a Systems Perspective," *SIGCOMM Comp. Comm. Rev.,* 38(3), 2008, pp. 75-78.

[8] D. S. De Couto, D. Aguayo, J. Bicket and R. Morris, "A High-Throughput Path Metric for Multi-Hop Wireless Routing," in *Proc. of the Int'l Conf. on Mobile Computing and Networking ,* 2003, pp. 134-146.