Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 52 (2015) 29 – 34

Procedia
Computer Science

The 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015)

Computations on the Edge in the Internet of Things

Andreas Moregård Haubenwaller[a], Konstantinos Vandikas[b]

[a] *Uppsala University, Uppsala, Sweden*
[b] *Management and Operations of Complex Systems, Ericsson Research, Stockholm, Sweden*

Abstract

In the Internet of Things (IoT), many applications focus on gathering data which can then be processed and visualized. However, such computations are usually spread generically based on parameters such as CPU and/or network load. This may mean that a significant amount of data needs to be transported over the network (either directly, or transparently using a network file system) in order for the data to be available to the node that is responsible for processing them. This paper proposes a method for deploying computations that can take factors such as data proximity into consideration. Thus, processing can be moved from central high-powered processing nodes to smaller devices on the edge of the network. By doing this, costs for gathering, processing and actuation can be minimized. In order to capture data dependencies among computations, but also to deploy and handle individual processing tasks in an easy way, the actor-model programming paradigm is used. To minimize the overall cost and to handle extra factors that weigh in on the distribution of tasks, a constraint programming approach is used. The combination of these two techniques results in an efficient distribution of tasks to processing resources in IoT. Taking into consideration the NP-hard nature of this problem, we present empirical results that illustrate how this technique performs in relation to the amount of devices/actors.

Keywords: iot; constraint-programming; actor model; edge computing

1. Introduction

By the year 2020, major technology companies expect that the number of connected devices will number in the range of 25-50 billion. Cisco and Ericsson believe that the number 50 billion devices will have been reached by 2020[1,2]. The Gartner Group on the other hand expects that number to be around 26 billion[3]. Despite the large disparity in numbers, they all see a huge increase in the number of connected devices. As more and more devices are being connected, new uses and markets are springing up. Some examples would be smart homes and home automation, environmental monitoring, and smart cities. All of these devices connected together coin the term ”Internet of Things”. Internet of things, or IoT, is currently getting a lot attention, for example it is on the peak of Gartner’s Hype Cycle

* Contact author. Tel.: +46 725398073
E-mail address: konstantinos.vandikas@ericsson.com

doi:10.1016/j.procs.2015.05.011

for 2014[4]. Along with this hype and attention, several platforms specifically aimed for gathering and visualizing IoT data were introduced, for example, SicsthSense[5], Xively[6], SensorCloud[7] and the IoT-Framework[8].

The main focus of these platforms is to gather data from a wide variety of devices found in IoT so that it can be analyzed, visualized and possibly acted upon. These devices can be anything from small wireless nodes to large servers. However, IoT devices as such are commonly seen as smaller and less capable than generic home computers or servers, some common examples would be; Raspberry Pi[9], Arduinos[10] and Intel Edison[11] boards. The normal approach is to gather data from all of the different devices by either a push(the devices initiate the sending of data) or a pull(the platform will ask the devices for data) approach. After data has been gathered it can be stored and analyzed by the platform.

This paper proposes an approach where the data will be processed by computationally capable IoT devices rather than being sent to a central location for processing. The processing tasks to be carried out, will be split up and deployed on these devices based on a certain cost. The goal is then to minimize this cost in order to find an efficient deployment of tasks on to devices. Depending on what cost metric is used, different costs can be minimized, for example one might want to reduce the overall amount of data, flowing through the network or one might strive to reduce response times. This will be accomplished by using a constraint programming model to minimize the overall cost while also being able to be extended with other constraints that may impact the search space of possible deployment alternatives.

2. Design

The idea behind this paper is to find a way to efficiently use the processing resources available in many devices connected to the Internet. Platforms in which you can register devices or data streams and then gather and visualize the data, already exist. The question is how to evolve such platforms in order to gain access to the processing capabilities of previously under-utilised computationally capable devices. Some of the aforementioned platforms might already support features such as triggers on data and actuation, however, processing is done centrally. By moving the processing to devices that are closer to the source of the data, the response times for actuation can be reduced. If the platform is using a pull type approach when it comes to gathering data, then some of that functionality could be moved to nearby devices so that data can be packed together and compressed before being sent which would reduce overall network traffic. If the platform is using a push type approach, then the destination of the data can be changed from the platform to a nearby device so that data can be packed and compressed.

2.1. Tasks and Task Graphs

Data flow can be seen as a task graph, where data originates from different devices; data can then be aggregated and possibly reduced in size via generalisation. Subsequently, data is sent along for further processing, storage and also possibly acted upon. When the data is acted upon, it would most likely cause actuation in a nearby device. If the central repository is far away, then the data has to travel a very long way; First it travels from the device with the sensor all the way to the repository where it is processed, and then it is returned the same way for the actuation. In order to get rid of this behavior, the data flow or task graph needs to be split up. Splitting the task graph up or rearranging is quite easy. However, splitting up complex software programs can be a daunting task. Chances are that these tasks are not represented as individual tasks at all in a large code base such as an IoT platform which would require quite a lot of code to be rewritten. Thus, if the program was already split up into smaller pieces then it would be a lot easier to ”break out” individual tasks and place them on different locations(devices). With this in mind, the actor based programming model[12] seems like a very good fit for this proposed IoT solution.

2.2. Cost-Based Distribution

In order to distribute the actors to devices in the Internet of Things one needs to figure out on which device each actor should be deployed. This will be solved by assigning a certain cost to all connections between actors as well as a cost between all devices. When an actor is deployed on a certain device, then the actor cost will be multiplied by the device cost. By deploying all actors on to devices, we can get a total cost of the actor graph. This cost can then be minimized by choosing different devices on which the actors are deployed on. The deployment should also be able to take others factors into consideration, such as the capabilities or the current load of the devices. The optimal solution would then be the solution with the minimal cost that does not break any other requirements.

In order to do the minimization of cost, a deployment algorithm is specified. The algorithm will try to find the optimal solution by minimizing the total cost of the distribution. The problem is to assign n actors on m devices. Several actors can be placed on the same device. The placement depends on the overall cost of the actor system. Each actor has a flow to all other actors and each device has the cost to communicate with all other devices. These costs can be seen as two matrices, the actormatrix $F = n * n$ and the devicematrix $D = m * m$. This problem closely resembles the well known Quadratic Assignment Problem(QAP) which is known to be NP-hard. Equation (1) shows the total cost of the deployment.

$$\sum_{i \in n} \sum_{j \in m} F(i, j) * D(i, j) \tag{1}$$

$$\sum_{a(i,j) \in A} F(i, j) * D(i, j) \tag{2}$$

Most actors will only have one or two connections to other actors so the F matrix will most likely be quite sparse. Therefore we model the cost by the sum of all of the connections or arcs between nodes. An arc $a(i, j)$ exists iff $F(i, j) \neq 0$. Let A denote the set of all arcs. The cost can now be expressed as Equation (2). In order to find the optimal solution, the cost seen in Equation (2) will have to be minimized. This can be done by deploying actors on different devices until the solution with the minimal cost is found.

3. Implementation

We implemented a very basic framework for handling the deployment of actors in Scala[13] and Akka[14]. The algorithm for finding the optimal deployment was also implemented in Scala with the use of the Scala toolkit OscaR[15]. Scala and the actor toolkit Akka was chosen as the language for a number of reasons. Mainly it is because it can run on the Java Virtual Machine (JVM); this provides the underlying infrastructure with a sort of delimiter on which devices that can actually be seen as processing resources. Any device that can run a JVM should be able to run actors.

The framework is written in such a way to receive a request for deployment from a user. It is assumed that the user has a set of actors of which are to be deployed. It is also assumed that the framework has access to a set of devices that are already registered and present in the system. The framework consists of three parts, the Deployer, the Scheduler and the Landmark. Each of these parts is an actor. The deployer is the actor that handles the deployment request from the user. The deployment request contains information about the actors - actorinfo. The landmark has information about the devices - deviceinfo. The landmark is a placeholder for where the underlying platform that has access to the registered devices would be.

3.1. Deployment Algorithm

The algorithm was implemented using constraint programming with the aid of the Scala toolkit OscaR[15]. Constraint programming was used as an approach because of the ease of adding and changing constraints that are not directly connected to the total cost of the distribution. The optimal deployment will be the deployment which has the lowest cost. However, by adding other constraints, such as static actors, the solution could be different. This has the added bonus of making the algorithm very flexible so that it is easy to add, remove and change constraints between deployment scenarios.

The goal of the algorithm is to assign n actors on m devices so that the overall cost is minimized. The cost between actors i and j is $F(i, j)$ and the cost between devices a and b is $D(a, b)$. Initially, no constraints are specified. This means that we have to check the whole search space in order to find the optimal solution. An array x of size n contains the decision variables. The decision variables can be seen as the tasks and they each contain a set of possible devices on which they can be deployed. The total cost of the sum, as can be seen in Equation (3), will be minimized using the branch and bound minimization which is found in OscaR[16].

$$\sum_{a(i,j) \in A} F(i, j) * D(x(i), x(j)) \tag{3}$$

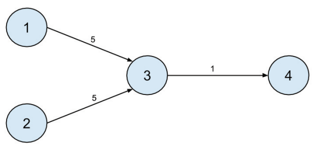

Fig. 1. The actorsystem

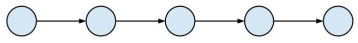

Fig. 2. The actors and their arcs in the best case

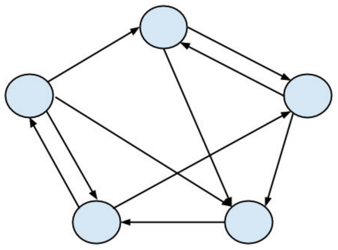

Fig. 3. The actors and their arcs in a worst case scenario

Possible constraints might be that some tasks can only exist on certain devices and also that some devices can only handle a certain number of tasks. For example: $x(y) = z, y \in n, z \in m$ would mean that task y must be placed on device z. Another example would be where each device can only hold one task, then we can put x under the alldifferent constraint.

3.2. Example of Deployment

In this example we have an actor system containing four actors and we have six devices. The actorsystem with the corresponding flows between actors can be seen in Figure 1 and the devices can be seen in Figure 4.
Actors one and two are producing data from sensors and they have to be placed on specific devices, namely devices one and two. Actor three will compress the data and send it to actor four that will store it. Actor four needs to be placed on a device that has access to the datastore, in this case that is device six or seven. The total cost of the deployment depends on the three arcs: (1,3), (2,3), (3,4). We know that actor one and two are placed on devices one and two, we also know that actor four must be placed on six or seven. That means that the domains of the actors are: Actor1 = {1}, Actor2 = {2}, Actor3 = {1,2,3,4,5,6,7} and Actor4 = {6,7}. We can now start the search by taking the first value in the domains and calculating the cost. In order to find the minimized cost, we need to check all solutions. The cost can now be expressed as seen in Equation (4).

$$F(1,3) * D(x(1), x(3)) + F(2,3) * D(x(2), x(3)) + F(3,4) * D(x(3), x(4)) \tag{4}$$

We know that x(1) = 1 and x(2) = 2. We also know the flow between actors, F(1,3) = 5, F(2,3) = 5 and F(3,4) = 1. The cost can now be simplified, see Equation (5).

$$5 * D(1, x(3)) + 5 * D(2, x(3)) + D(x(3), x(4)) \tag{5}$$

The latency matrix D can be seen in Table 1. The possible solutions and the cost for each solution can be seen in Table 2.
From the results, we can see that we have two solutions with the lowest cost, which means that we have two optimal solutions. These solutions would be: Actor1={1}, Actor2={2}, Actor3={1}, Actor4={6} (See Figure 5) OR Actor1=1}, Actor2={2}, Actor3={2} and Actor4={7} (See Figure 6)

4. Scalability Test

It is interesting to see how many actors and how many devices the algorithm can actually handle due to the NP-hard nature of the problem. This test tries different numbers of actors and devices and measures the time it takes for algorithm to find the optimal solution. The timeout for this test is set at one(1) hour. The common criteria for the tests are that the first and the last actors are static and, that the costs of the arcs between actors and the costs of the links between devices are random between 1-100. The number of arcs between actors and how they are connected is what distinguishes the best case from the worst case. In the best case, each actor has an arc to the next one, except for the last actor(see Figure 2). In the worst case, each actor has an arc to the next actor and the last actor has an arc

Table 1. The latency matrix D

Device	1	2	3	4	5	6	7
1	0	20	20	20	40	50	70
2	20	0	40	20	20	70	50
3	20	40	0	20	50	30	50
4	20	20	20	0	20	30	30
5	40	20	40	20	0	50	30
6	50	70	30	30	50	0	20
7	0	0	50	30	30	20	0

Table 2. Possible solutions for the deployment

Actor1	Actor2	Actor3	Actor4	TotalCost
1	2	1	6	150
1	2	2	6	170
1	2	3	6	330
1	2	4	6	230
1	2	5	6	350
1	2	6	6	600
1	2	7	6	620
1	2	1	7	170
1	2	2	7	150
1	2	3	7	350
1	2	4	7	230
1	2	5	7	330
1	2	6	7	620
1	2	7	7	600

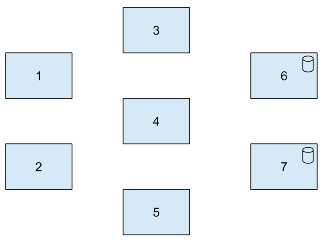

Fig. 4. Available devices capable of processing

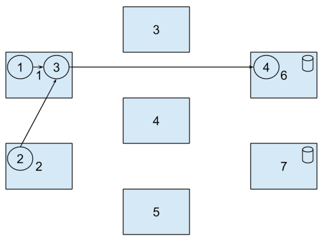

Fig. 5. Deployment 1

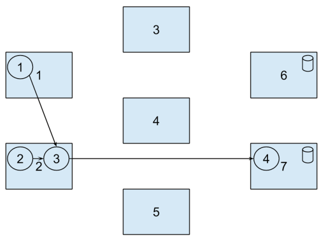

Fig. 6. Deployment 2

to the first one; This means that there is a circular flow. In addition to this, there are arcs from random actors to other random actors(see Figure 3). In total, the number of arcs in the worst case is the number of actors times two, whereas in the best case the number of arcs is the number of actors minus one.

The tests were performed on an Intel Core i5-3570K @ 3.40GHz with 8 GB of RAM running Windows 8.1. The results indicate that for smaller numbers of actors, the optimal deployment can be found very fast. However, as the number of actors grows, the time it takes to find an optimal solution increases dramatically. The fact that the algorithm might take 10 minutes to find a deployment is still a reasonable amount of time, in particular in non mission critical scenarios, since the actors might be deployed for days, weeks or even years. However, when the algorithm times out, the time it would take to find a deployment is just too long. In such cases, one might want to change the algorithm and try some other heuristics so that an approximate solution is found instead. This would mean that the deployment is not guaranteed to be optimal, but it should still provide a reasonably good deployment. Detailed times for best case/worst case scenario are shown in Tables 3 and 4 where the rows indicate the different amount of devices and the columns the different amount of actors used for each test.

5. Conclusions and Future Work

In this paper, a new approach for handling data and using processing resources in the Internet of Things is proposed. Most of the IoT platforms today only perceive the devices in the Internet of Things as producers of data; this is something that needs to change if the number of connected devices will follow the trend that is seen by companies such as Ericsson and Cisco. Not only that, but the processing capabilities of these devices are generally overlooked. By using actors and a cost-based approach, the processing capabilities of these devices can be utilized and this could lead to less traffic being sent over the network, especially over expensive links, and also a significant reduction in

Table 3. Best Case:

	5	10	15	20	25
5	13 ms	4 ms	3 ms	13 ms	1.067 s
10	12 ms	14 ms	20 ms	12 ms	6.265 s
15	14 ms	32 ms	86 ms	138 ms	2.244 s
20	20 ms	30 ms	104 ms	168 ms	4.709 s
30	10 ms	67 ms	135 ms	683 ms	88.040 s
50	21 ms	1.062 s	4.748 s	14.734 s	≈10 mins
100	38 ms	104 ms	3.913 s	>1 h	≈11 mins

Table 4. Worst Case:

	5	10	15	20	25
5	42 ms	8 ms	29 ms	89 ms	11.244 s
10	32 ms	239 ms	61 ms	296 ms	≈7 mins
15	7 ms	119 ms	2.544 s	≈2 mins	>1 h
20	4 ms	532 ms	2.940 s	82.897 s	>1 h
30	15 ms	1.151 s	1.708 s	>1 h	>1 h
50	90 ms	2.787 s	22.480 s	39.842 s	>1 h
100	81 ms	≈9 mins	>1 h	>1 h	>1 h

response times for certain deployments. Using a cost-based deployment method to factor in data proximity and cost of links can potentially save a lot of money in network costs and at the same time reduce response times.

One of the limitations described in the aforementioned approach is that the deployment that has been generated cannot be changed once it is deployed. Consequently, it is interesting to investigate more flexible deployment techniques which allow for transferring on the fly one actor from one device to another, taking into consideration the current state of the actor and available CPU and memory resources on each device. This on-the-fly deployment would also be very useful in the case of a faulty device. If a device goes down, then it could possibly interrupt the flow of data. In such a case, the deployed actor system could heal itself by redeploying or moving the actor to a functional nearby device.

Moreover, it is interesting to investigate possible optimizations in the constraint-programming algorithm such as randomization and heuristic approaches in order to minimize the search space of the algorithm and find near-optimal solutions in smaller amount of time.

Acknowledgements

This work is supported by the EU FP7 CityPulse Project under grant No.603095.

References

1. Ericsson, . More than 50 billion connected devices - taking connected devices to mass market and profitability. White Paper; Ericsson; 2011.
2. Evans, D.. The Internet of Things How the Next Evolution of the Internet Is Changing Everything. White Paper; Cisco Internet Business Solutions Group (IBSG); 2011.
3. The Gartner Group, . Gartner says a thirty-fold increase in internet-connected physical devices by 2020 will significantly alter how the supply chain operates. Press Release; 2014. Available from `http://www.gartner.com/newsroom/id/2688717`.
4. The Gartner Group, . Gartner's 2014 hype cycle for emerging technologies maps the journey to digital business. Press Release; 2014. Available from `http://www.gartner.com/newsroom/id/2819918`.
5. SICS, . Sicsthsense. 2014. Available from `http://sense.sics.se/`.
6. LogMeIn, . Xively. 2014. Available from `https://xively.com/`.
7. SensorCloud, . Sensorcloud. 2014. Available from `http://sensorcloud.com/`.
8. Vandikas, K.. IoT-Framework. 2014. Available from `http://http://www.ericsson.com/research-blog/internet-of-things/computational-engine-internet-things/`.
9. Raspberry Pi Foundation, . Raspberry Pi. 2014. Available from `http://www.raspberrypi.org/`.
10. Arduino, . Arduino. 2014. Available from `http://www.arduino.cc/`.
11. Intel, . Intel Edison. 2014. Available from `http://www.intel.com/content/www/us/en/do-it-yourself/edison.html`.
12. Hewitt, C., Bishop, P., Steiger, R.. A universal modular actor formalism for artificial intelligence. In: *Proceedings of the 3rd international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc.; 1973, p. 235–245.
13. Odersky, M.. Scala - Scalable Language. 2014. Available from `http://www.scala-lang.org/`.
14. Typesafe Inc., . Akka. 2014. Available from `http://akka.io`.
15. OscaR Team, . OscaR: Scala in OR. 2012. Available from `https://bitbucket.org/oscarlib/oscar`.
16. Pierre Schaus, . CP for the impatient. 2013. Available from `http://info.ucl.ac.be/~pschaus/cp4impatient/firststeps.html`.