Construction of computing balancing model in the Internet of Things devices system

Aleksandr Sidorov, Sergey Slastnikov
National Research Institute High School of Economics, Moscow, Russia

E-mails: asidorov@hse.ru, sslastnikov@hse.ru

Abstract: This article is dedicated to our work in field of research and development of a math model for load balancing in the Internet of things (IoT). Here, we perform analysis and classification of tasks in the IoT devices system. We split subprograms in following systems on balancing and non-balancing. Next, we classify balancing subprograms by different parameters. Then, based on those classifications, we construct model for load balancing in IoT. With that model, in future we can perform modeling IoT devices systems and found pros and contras of each balancing algorithm.

1. Introduction
At this work, we start to research development and construction of a math model for load balancing in the Internet of Things devices system. Suggested model is useful to perform experiments of modeling the parameters and behavior of the IoT devices system with different scripts of data loading. The main goals of current work are to reduce delays in the performance of tasks, to increase battery life and energy efficiency. The objectives are to describe tasks performed in IoT systems, to summarize definitions of tasks balancing methods, to develop a math model in the IoT devices system and to design various balancing methods for this model.

At first, we decide to perform analysis and classification of tasks for the IoT devices system. Based on this classification, we assume that not each kind of tasks (and associated computations) could be reassigned to another node in system.

At second part, we compare existing methods of computing balancing for IoT systems. We classify methods by its centralization, scaling and flexibility. Here we introduce our balancing method based on the computation offloading at heterogeneous systems with different levels of performance. This method allows moving computation to any performance level node. There are some works of other authors [1] [2], describing similar methods, but mostly concentrated on transfer tasks to higher performance level (e.g. cloud center).

At the end, we develop the mathematical model of the Internet of Things devices system. It is particularly based on the model proposed in work by Elizarov M. [3]. Model extended for balancing processes and node health check.

In advance, results of this work could be used for modeling and comparative analysis of balancing methods. Further, we will compare obtained results with other results, based on experiments on the virtual and physical system of devices to determine the quality of the constructed model.
2. Classification of calculations
At the beginning, we classify calculations, performed in considered systems and its balancing methods. Under term “calculation”, we mean processing methods (as Object-Oriented Programming term) and subprograms. In real usage every subprogram in Internet of Things devices system unites in computation chain with others. This chain is called – task.

Our main goal in this section is assign calculations in two groups – balancing and non-balancing, and find their distinguish parameters.

At first, we will describe some rules that are necessary for computing balancing.
1. Runtime environment equality. Subprogram may be balanced between nodes A and B if subprogram is executable on these nodes (hardware and software).
2. Execution equality. Subprogram may be balanced between nodes A and B if result of execution (output parameters) will be equal at both nodes.
3. Data availability. Subprogram may be balanced between nodes A and B, working with data on node C if from both nodes A and B subprogram could access (read or write) to the node C data.

Consider subprogram assignment by following parameters:
• function;
• connectivity.

Each program in task performs specific use, for example:
• collecting data from sensors;
• transmitting control signals to actuators (for example, electric drives);
• data’s zip and unzip;
• nodes sync and etc.

We can develop uncountable set of subprograms with different usage. In addition, one subprogram can unite usages of two different subprograms, for example, message-sending subprogram can zip transmitted data.

However, we can select type of local subprogram. These local subprograms use local physical node’s parameters as input or output parameters (reading sensors, transmitter, self-diagnose subprogram, etc.). Obvious that these types of subprograms couldn’t be balance, because results on different nodes differ or their execution fail.

Subprogram connectivity describes by intensiveness of data exchange between neighbor subprograms in task. The more intensiveness of data exchange goes to the more connectivity between tasks. Obvious, that subprogram with high-level intensiveness should be performed on one node. It could reduce network load. At the following works, we are going to find connectivity level threshold, after that we should execute them on one node.

Therefore, we can select subprograms with high-level connectivity and local subprograms as non-balanced subprograms. Non-local subprograms with low-level connectivity may be rebalanced when higher described rules are respected. Balancing is allowed if it leads to improvement of system’s quality parameters – decrease net load, summary task execution time, nodes load, etc.

Although calculations perform in method or subprogram, we can divide subprograms on two types:
• movable;
• fixed.

Movable subprograms can move their source code between nodes in net and execute on other system’s nodes. These subprograms does not require their copy on the receiver node and can copy their source code to receiver node while performing balancing. However, this type may have difficulties with capability between source code and hardware on receiving node, which cause problems in heterogeneous systems for compiled subprograms. Solution for this problem is in using interpreted languages or virtual machines but these cause higher load on memory and CPU and may increase task’s execution time.

Fixed subprograms can’t move their source code. For this type of subprograms calls redirection between nodes performs while calculations balancing. It reduces balancing time because we don’t
need to copy subprogram source. These subprograms are preinstalled on nodes, so they can be optimized for node’s hardware, improving CPU load and performance. Lack of fixed subprograms is that calculations can be moved only to the node where this or similar subprogram is installed. Also, node store every subprogram even if its unused, which cause higher memory use.

3. **Classification of balancing methods**
Different algorithms can balance subprograms. For cloud computing great taxonomy of resource allocation is defined in [4]. We accept this taxonomy for Internet of Things systems:

- By balancing order - we can split methods on preventive, on detect and mixed. Preventive methods perform tasks distribution once at their start moment. “On detect” algorithm performs tasks redistribution on detection of disbalance or overload.Mixed method uses both sides, and as result, it requires more resources than both.
- By search area - we can split methods on global and local. Global balancing methods are trying to find optimal solution using all nodes in the system. Local methods try to improve current system’s configuration using limited count of surrounded nodes to the origin. Obvious that local methods take less time for perform balancing. On the other side, in general local search does not get adorable results.
- By relocation count - we split methods on single and group. Single relocation methods are aimed to reduce specific node’s load. They start after critical overload on one of the optimized parameters. Group methods try to achieve optimal load between nodes. They trigger when occurs critical difference between minimum and maximum nodes’ load in system.

Also, we should highlight static tasks allocation (STA). STA appears when developer or administrator distributes tasks to nodes manually while adding. It’s effectivity depends on developer skills, but it is much more complicated with huge task amount. Obvious, that STA doesn’t use background tasks for load control and rebalance, so higher effectivity of load using is achieved.

This classification doesn’t define how algorithms decide which subprograms are going to balance. For example, in [5] and [6] authors used fuzzy logic. Various decision algorithms achieve different effectiveness. In future work, we will compare different decision algorithms such a fuzzy logic, least connected, most CPU usage etc.

In Table 1 we define algorithms for each combination of parameters. STA is defined as $a_0$.

|      | Preventive | On detect | Mixed |
|------|------------|-----------|-------|
|      | Single     | Group     | Single | Group |
| Global | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ |
| Fixed | $a_7$ | $a_8$ | $a_9$ | $a_{10}$ | $a_{11}$ | $a_{12}$ |
| Local | $a_{13}$ | $a_{14}$ | $a_{15}$ | $a_{16}$ | $a_{17}$ | $a_{18}$ |
| Fixed | $a_{19}$ | $a_{20}$ | $a_{21}$ | $a_{22}$ | $a_{23}$ | $a_{24}$ |

4. **Model of IoT system’s for calculations’ balancing**
Here we describe model for IoT devices system:

1. Each subprogram is described as {memory usage, operations count, input parameters, output parameters, trigger event, execution node}
   
   \[ SubP = \{M, Op, In, Out, trigger, node\} \]

2. Each task is ordered set of subprograms
   
   \[ Task = \{SubP_1 ... SubP_n\} \]

3. Subprograms are connected through output parameters and input parameters or trigger event
4. Nodes performance is count of operations per time and memory value

\[ Node.p = \{OPS, M\} \]

5. Balancer – special task that can move subprograms among the nodes

\[ TaskBalancer = SubP\_node[v_1 \rightarrow v_2] \]

6. Network interconnection is based on model by M. Elizarov [3].

7. Balancer execution time consist of:
   a. \( t_{discover} \) – time, that balancer needs to find disbalance from time it appears;
   b. \( t_{search} \) – time, that balancer needs to find receiver node;
   c. \( t_{move} \) – time needed for copy source codes of subprogram;
   d. \( t_{reconnect} \) – time needed for restore connections with relocated subprogram;

8. System’s state is scored by following parameters:
   a. By nodes’ load:
      i. On CPU:
         1. \( CPU\_load_{Me} \) – median CPU load of node;
         2. \( CPU\_load_{max} \) – maximum CPU load of node;
      ii. On memory:
         1. \( MEM\_load_{Me} \) – median memory load of node;
         2. \( MEM\_load_{max} \) – maximum memory load of node;
      iii. On network:
         1. \( Net\_load_{Me} \) – median network load;
         2. \( Net\_load_{max} \) – maximum network load.
   b. By tasks’ execution time:
      i. \( t_{diff}_{Me} \) – median difference between real time of tasks’ execution time and reference execution time;
      ii. \( t_{diff}_{max} \) – maximum difference between real time of tasks’ execution time and reference execution time.

Reference execution time – task’s execution time in the system without other tasks (as it was only one executed in the system).

At first stage of modelling we suggest that nodes have stable connections with each other and available bandwidth is much higher than actual data rates between nodes, i.e. network errors’ influence is minimal.

5. Summary

In this article we perform classification of subprograms in Internet of Things to select computations that available for balancing. This classification doesn’t performed yet.

We select parameters of calculation balancing methods. These parameters are close to taxonomy for cloud computing balancing. By these parameters we can find balancer execution time easier because of their classification.

We describe model for IoT devices systems computing balancing. We decide to use median scoring instead of average as more objective score.

In following work, we continue to research balancing methods. We will find formulas for execution time of computing balancing, create configurations for modeling systems, analyse results and create experiment with virtual machines to estimate model.
References
1. Flores H., Xiang S., Kostakos V., Yi Ding A., Nurmi P., Tarkoma S., Hui P., Li Y. 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops) // Large-scale offloading in the Internet of Things. Kona, HI, USA. 2017.

2. Samie F., Tsoutsouras V., Bauer L., Xydis S., Soudris D., Henkel J. 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT) Reston, VA, USA. 2016. pp. Computation offloading and resource allocation for low-power IoT edge devices.

3. Elizarov M. MODELS AND ALGORITHMS OF INFORMATION INTERACTION IN THE INTERNET OF THINGS NETWORKS. Saint-Petersburg: 2017.

4. Mustafa S., Nazir B., Hayat A., Khan A.U.R., Madani S.A. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems // Comput. Electr. Eng. Oct 2015. pp. 186–203.

5. Hatti D.I., Sutagundar A.V. International Conference on Recent Advances in Electronics and Communication Technology // Agent Based Job Classification and Resource Allocation in IoT. 2017.

6. Sutagundar A.V., Manvi S.S. The Third Advanced International Conference on Telecommunications // Agent Based Approach to Information Fusion in Wireless Sensor Networks. 2017.