Synthesis of real-time cloud applications for Internet of Things

Sławomir BAK¹, Radosław CZARNECKI¹, Stanisław DENIZIAK¹,²,*
¹Department of Computer Engineering, Cracow University of Technology, Krakow, Poland
²Department of Computer Science, Kielce University of Technology, Kielce, Poland

Abstract: This paper presents the methodology for the synthesis of real-time applications working in the “Internet of Things” environment. We propose the client-server architecture, where embedded systems act as smart clients and the Internet application is a server of the system. The architecture of the application conforms to the cloud computing model. Since centralized systems are prone to bottlenecks caused by accumulation of transmissions or computations, we propose the distributed architecture of the server and the methodology that constructs this architecture using Internet resources supported by a cloud provider. We assume that the function of the server is specified as a set of distributed algorithms, and then our methodology schedules all tasks on the available network infrastructure. It takes into account limited bandwidth of communication channels as well as the limited computation power of server nodes. The method minimizes the cost of using network resources that are necessary to execute all tasks in real-time. We also present a sample application for adaptive control of traffic in a smart city, which shows the benefits of using our methodology.

Key words: Internet of Things, cloud computing, real-time system, system synthesis, embedded systems

1. Introduction

The Internet of Things (IoT) is a concept in which the real world of things is integrated with Internet technologies [1]. More and more Internet-enabled devices are now available (mobile phones, smart TVs, navigation systems, tablets, etc.). It is expected that, in a few years, almost every product may be identified and traced on the Internet using radio frequency identification (RFID), near field communication, or other wireless communication methods. This will make possible the development of a huge number of Internet applications. However, the enormous growth of devices connected to the Internet may cause additional problems relative to the computational complexity of IoT-based systems and communication bottlenecks. Existing Internet technologies may be insufficient for dealing with IoT systems; thus, new web architectures, communication technologies, and design methods should be developed to enable development of efficient IoT systems [2–4].

One of the most interesting domains is the Internet of Smart Things, where IP-connected embedded systems built in our environment may interact with the Internet. Smart devices not only incorporate sensing/monitoring and control capabilities, but also may cooperate with other devices and with Internet applications. For example, an adaptive car navigation system may interact with an Internet system, controlling and monitoring the traffic in a city to avoid traffic jams. In such a case, the Internet acts as a distributed server that processes requests sent by smart devices implementing client applications. Usually responses to the device should be sent during a limited time period. Therefore, this class of IoT application is a real-time system.
Current work concerning the IoT mainly concentrates on 3 domains: developing new wireless communication technologies and protocols, adopting existing web technologies to this new paradigm, and developing software architectures for IoT applications. Besides low-range communication systems based on RFID, such as RFID sensor networks [5] or wireless identification and sensing platforms [6], IP-based communication systems were also developed. 6LoWPAN [7] or Internet of Things [8] approaches implement low-power light IP protocols that may be built in almost any object; therefore, the vision that anything may be addressed and reachable from any location may be realistic in the near future. Communication systems define the lowest layer of the IoT system, and upon this layer the software layers consisting of middleware and applications are defined. Middleware defines the software architecture and it is composed of a set of Internet and web technologies. Most approaches adopt existing web technologies to IoT applications. Usually architectures based on service-oriented architecture or representational state transfer [9] are applied. The integration of smart things with the web is also called the Web of Things (WoT).

In [3], the problem of real-time requirements in WoT applications was discussed. The authors observed that many WoT systems interact with embedded devices and expect real-time data, and thus the development of WoT applications that satisfy real-time requirements is one of the main challenges. Although some technologies for real-time communication (e.g., RTP/RTSP, XMPP) or real-time interaction (e.g., Comet [10]) have been developed, still more developments and standards are required for real-time WoT and IoT systems.

Distributed Internet applications require an expensive network platform consisting of servers, routers, switches, communication links, etc. to operate. The cost of the system may be reduced by sharing the network infrastructure between different applications. This is possible using the infrastructure as a service (IaaS) model [11] of the cloud computing services [12]. IaaS together with a real-time cloud environment [13] seems to be the ideal platform for IoT systems. However, to guarantee the quality of service and minimize the cost of the system, efficient methods of mapping IoT applications onto network infrastructure should be developed.

One of the previous methods selecting resources from a cloud is based on the concept of game theory [14]. The method optimizes cost and performance. This concept reflects the common characteristics of the physical position and bandwidth available between the job and the resources, and it emphasizes establishing a scheduling relationship between near entities. In resource scheduling, the choice of a near and low-cost resource is a key criterion. Zhi et al. [15] described the scheduling algorithm and protocol. In this cost-based method, the set of computing resources with the lowest price are assigned to the user according to current resource availability and price. Another method proposed scheduling resources based on a genetic algorithm [16]. In this method, the scheduling scheme is coded in an integer sequence and a fitness function is based on influence degree. The genetic operations include selection, crossover, mutation, and elitist selection. None of these methods consider resources shared between different reactive real-time applications.

In this paper, we present the methodology for the synthesis of reactive, real-time cloud applications that are in accordance with the IoT concept. We assume that each thing is represented as an embedded system that may send requests to Internet applications and expects the response in the specified time. The goal of our methodology is to find the distributed architecture of the application that will satisfy all user requirements. The method also minimizes the cost of IaaS services required for running the IoT application in the cloud environment.

The next section presents our assumptions and the definition of the real-time Internet of things (RTIoT) application. In Section 3, the method of synthesis will be described. Section 4 presents an example and experimental results demonstrating the advantages of the methodology. The paper ends with conclusions.
2. Problem statement

System synthesis is a process of automatic generation of the system architecture, starting from the formal specification of functional and nonfunctional requirements. The functional requirements define functions that should be implemented in the target system. Nonfunctional requirements usually define constraints that should be fulfilled, e.g., time constraints define the maximal time for execution of the given operations and cost requirements define the maximal cost of the system.

The functions of distributed systems are usually specified as a set of communicating tasks or processes. We use a similar model for the specification of IoT applications, the details of which are given in Section 2.1. We consider real-time systems; hence, the main set of requirements concerns time constraints. IoT uses existing network infrastructure consisting of servers, routers, and connections. Therefore, the target architecture of the IoT system should fit into the available network architecture, but it may not guarantee that all time requirements will be met. In this case, the infrastructure should be extended by adding some components, i.e. additional resources should be hired from cloud providers. Thus, it should be possible to specify architectural requirements that have to be satisfied by the target system. The model of the target architecture is described in Section 2.2, and requirements that are used in our methodology are presented in Section 2.3.

2.1. Functional specification

The reactive IoT system should be able to process thousands of requests in a short time period. This will be possible only if massive parallel computing is applied. Therefore, the functional specification of the system should represent the function as a distributed algorithm [17] developed according to the following requirements:

1. **Parallel model of computations**: The system should be specified as a set of parallel processes using message passing communication.

2. **Parallel request handling**: A huge number of requests may cause a communication bottleneck; to avoid this, simultaneous requests should be handled by different processes.

We assume that the system is specified as a collection of sequential processes coordinating their activities by sending messages. Specification is represented by a graph $G = \{V, E\}$, where $V$ is a set of nodes corresponding to the processes and $E$ is a set of edges. Edges exist only between nodes corresponding to communicating processes. Tasks are activated when the required set of events appears. As a result, the task may generate other events. External input events will be called requests ($Q$), external output events will be called responses ($O$), and internal events correspond to messages ($M$). The function of the system is specified as finite sequences of activation of processes. There is a finite set of all possible events $\Lambda = Q \cup O \cup M = \{\lambda_i : i = 1, ..., r\}$. For each event $\lambda_i$, communication workload $w(\lambda_i)$ is given. System activity is defined by the following function:

$$\Phi : C \times V \rightarrow \omega \times 2^\Lambda,$$

where $C$ is an event expression (logical expression consisting of logical operators and Boolean variables representing events) and $\omega$ is the workload of the activated process.

Using function $\Phi$ it is possible to specify various classes of distributed algorithms. Figure 1 presents a sample echo algorithm [18] consisting of 5 processes. The algorithm consists of 10 actions. Each action is activated only once, when the corresponding condition equals true. All actions except $A_1$ and $A_6$ contain alternative subactions. Only the first action for which the condition is satisfied will be activated.
the echo algorithm specification, process \( v_1 \) is the initiator (i.e. is activated first), messages \( m_1, \ldots, m_{14} \) are explorer messages (i.e. messages propagating request), and \( m_{15}, \ldots, m_{25} \) are echo messages (i.e. messages propagating responses). Indices are added only for readability, and \( m_x \) means that message \( m_x \) is associated with edge \( e_x \) in the graph. For the same reason, edge names in the event expressions mean any received message corresponding to this edge, e.g., \( e_1 = m_1 \mid m_4 \mid m_{15}, e_2 = m_2 \mid m_{9} \mid m_{20} \). Events \( x_1, \ldots, x_{11} \) are internal events used for storing the state of processes between successive executions.

![Diagram of the echo algorithm](image)

**Figure 1.** Sample specification of the echo algorithm.

Since different requests may be processed by distinct algorithms, the function of a system may be specified using a set of functions \( \Phi \) sharing the same processes. Each function has only one initiator (process activated by the request). Processes may be activated many times, but the algorithm should consist of the finite number of actions and infinite loops are not allowed.

### 2.2. Target architecture

The proposed architecture of the RTIoT system is composed of four layers: the Things Layer (QL), the Wireless Layer (WL), the Network Layer (NL), and the Server Layer (SL). The dependencies between layers are shown in Figure 2.

![Diagram of the RTIoT system](image)

**Figure 2.** Layers in the RTIoT system.
Layer QL consists of embedded systems (smart things) managed by the RTIoT. We assume that things are mobile systems \((q_i)\), and thus the number of things \(N_q\) may change in time but is limited to \(Q_{\text{max}}\). Let \(Q(t_i) = \{q_j; j = 1, \ldots, N_q(t_i)\}\) be the set of things available in the RTIoT system in timeframe \(t_i\). Each \(q_x\) has a connection with the nearest access point \(AP\) in layer WL, i.e.:

\[
\forall t_i \forall q_j \exists AP^j_k : q_j \rightarrow AP_k,
\]

where \(t_i\) is the time frame, \(AP^j_k\) is the access point nearest to \(q_j\) at time \(t_i\), and \(\rightarrow\) means the connection. Since \(q_j\) is a mobile system, it may change locations in time. This may cause the nearest access point to also be changed in the following timeframes; thus, for some time frames, it may appear that \(AP^j_k \neq AP^j_{k+1}\).

Layer WL consists of access points (AP). The access points should guarantee that each \(q_i\) can communicate with the system at any time. Therefore, all APs should cover all possible geographical locations of \(q_i\) and the number of APs is constant. For each \(AP_i\), we may identify the maximal number of \(q\)s that may appear in the region covered by this \(AP_i\). Each \(AP_i\) is permanently connected to just one router \(R_j\) from the network layer (NL). We assume that layer WL will also satisfy the following condition:

\[
\forall AP_i : \sum_j B(q^j_i) \leq B(AP_i),
\]

where \(B(q^j_i)\) is the peak transmission rate for \(q_j\) connected to \(AP_i\), and \(B(AP_i)\) is the bandwidth of the communication link between \(AP_i\) and a router.

Layer NL consists of routers (Rs) and communication links (CLs). For each \(CL_i\) the available bandwidth \(B(CL_i)\) is defined. Each router may be connected with any number of access points, with other routers, and with servers.

Layer SL contains servers (Ss) consisting of compute nodes \(N_i\). Each \(N_i\) is characterized by performance \(P_i\) reserved for the RTIoT system, and it may be equipped with a network interface. Thus, each compute node may be connected to another router.

The goal of our methodology is to find the cheapest system architecture for an application that fulfills all time constraints and uses the existing network infrastructure available in a cloud. All servers (nodes) and communication resources that are used in the target architecture \(\Pi_T = \{S, R, AP, CL\}\) of the system are outsourced to the cloud provider. The method starts from the initial architecture \(\Pi_I = \{S', R', AP', CL'\}\) consisting of the fastest resources. Next, the architecture is optimized by performing some modifications of \(\Pi_I\) where only resources supported by cloud providers are considered. Our methodology minimizes the cost of hiring the network infrastructure by achieving the maximal utilization of all resources and by allocating the cheapest components that satisfy all time constraints. Each available resource is characterized by properties defining the performance and the cost of the corresponding IaaS service. Specifications of all available resources constitute the database of resources \(L = \{R'', CL'', S''\}\).

\(R''\) is a set of available routers. Each router \(r_i \in R''\) is defined by the following properties:

- \(Cr(r_i)\) - cost of the routing service;
- \(n_p(r_i)\) - the number of ports;
- \(P(r_i)\) - set of ports, where for each port \(p_i\) \((i = 1, \ldots, n_p(r_i))\) the maximum available bandwidth of the port \((B(p_i))\) is defined.
Communication links $cl_i \in CL''$ are characterized by the maximal available bandwidth $B(cl_i)$, bandwidth $B_r(cl_i)$ reserved for the application, and price of the communication service $Cr(cl_i)$ for each available bandwidth. The communication link connects any router port with a port of another router, an access point port, or a network interface port. The final bandwidth for such connection $c_j$ is defined as:

$$B(c_j) = \min(B(p_s), B_r(cl_i), B(p_d)),$$

(4)

where $p_s$ and $p_d$ are ports connected by communication link $cl_i$. Thus, the time of transmission of packet $D_i$ through connection $c_j$ is the following:

$$T(D_i) = \frac{l(D_i)}{B(c_j)},$$

(5)

where $l(D_i)$ is the length of packet $D_i$.

We assume that each server $s_i$ is configurable and may consist of any number of nodes, i.e. a multiprocessor or cluster architecture of the server. Each node may execute all assigned tasks sequentially. Thus, the following properties characterize the server:

- $n_s(s_i)$ - Number of nodes; hence, server $s_i$ may be represented as a set $\{N_1(s_i), \ldots, N_{n_s}(s_i)\}$ of nodes.

- $Cr(s_i)$ - Cost of the computing services; the cost depends on the number of nodes allocated to the application, and usually the cost function is not linear. A sample cost function for a cluster server is presented in Figure 3.

- $\{P_1(s_i), \ldots, P_{n_s}(s_i)\}$ - Performance of each node.

- $\{B_1(s_i), \ldots, B_{n_s}(s_i)\}$ - Bandwidth of each network interface.

![Figure 3. Sample cost function for the cluster server.](image)

The time required for executing process $\tau_i$ by node $N_j(s_i)$ equals:

$$T(\tau_i) = \frac{w(\tau_i)}{P_j(s_i)},$$

(6)

where $w(\tau_i)$ is the workload of task $\tau_i$.

Figure 4 presents a sample target architecture of a RTIoT system.
2.3. Requirements and constraints

Let \( \rho(\lambda_x, \lambda_y) \) be a sequence of actions \( A_1, \ldots, A_s \) such that \( \lambda_x \) is the request, \( \lambda_y \) is the response, and:

\[
A_1 : \Phi(v_1, \lambda_x) \rightarrow \{ \omega_1, M_1 \}, A_s : \Phi(v_s, \lambda_y) \rightarrow \{ \omega_s, \{ \lambda_y \} \}, \quad \forall 1 < k < s - 1 \quad A_k \rightarrow A_{k+1},
\]

(7)

where \( v_i, v_j \) are any processes, \( M_1 \) is a set of messages sent after finishing action \( A_1 \), and \( A_k \rightarrow A_{k+1} \) means that action \( A_k \) generates events activating action \( A_{k+1} \). The time of execution of the given sequence of actions is then defined as the sum of the following times: time required for transmitting the request, execution times of all processes, time of interprocess communication, and transmission time of response:

\[
t(\rho(\lambda_x, \lambda_y)) = \frac{\omega(\lambda_x)}{B_r(\lambda_x)} + \sum_{i=1}^{s} \frac{\omega(A_i)}{P(A_i)} + \sum_{i=1}^{s} \frac{\omega(m_i)}{B_r(m_i)},
\]

(8)

where \( \omega(A_i) \) is the workload of the process activated by action \( A_i \), \( P(A_i) \) is the performance of the server executing this process, \( \omega(m_i) \) is the communication size, and \( B_r(m_i) \) and \( B_r(\lambda_y) \) are reserved bandwidths of channels used for sending the message or request. If processes activated by actions \( A_k \) and \( A_{k+1} \) are executed by the same server, then \( \omega(m_k) = 0 \) for any message sent between these processes.

The time constraint is the maximal period of time that may elapse between sending a request by the smart thing and receiving the response. Since the request may activate different sequences of actions until the response is obtained, the time constraint (deadline) is defined as:

\[
t_{\text{max}}(\lambda_x, \lambda_y) = \text{MAX}(t(\rho(\lambda_x, \lambda_y))).
\]

(9)
During the synthesis, processes and transmissions are scheduled and assigned to network resources. The method first assigns processes and transmissions to the fastest resources supported by cloud providers. This step verifies whether or not it is possible to find the network infrastructure that fulfills all time constraints. Next, the cost is minimized by performing the following modifications:

- Decrease communication bandwidth $B_{cl_i}$ for any allocated communication channel; in this way, the cost of communication service $Cr(cl_i)$ may be reduced.
- Remove router or communication links; in this way, the cost of routing services $Cr(r_i)$ will be reduced.
- Change server $s_i$ to a cheaper one or reduce the number of allocated nodes; in this way, the cost of computing services $Cr(s_i)$ will be reduced.

Only modifications that do not violate time constraints are considered. The optimization process will stop when each considered modification of the architecture causes a violation of time requirements. Hence, the total cost of the IaaS service is the following:

$$C_M = \sum_i Cr(cl_i) + \sum_j Cr(s_j) + \sum_k Cr(r_k).$$

The goal of our methodology is to minimize $C_M$.

3. Synthesis

Our method of synthesis starts from the formal specification of the RTIoT system (as described in Section 2.1) and tries to produce the cheapest target architecture of the system that satisfies all constraints. The method minimizes the cost of outsourcing the network infrastructure to the IaaS-cloud provider.

3.1. Assumptions

The method is based on the worst-case design. We assume that the workload of each action and the sizes of all transmissions are estimated for the worst cases. All time constraints should also satisfy the following condition:

$$t_{max}(\lambda_q, \lambda_o) \leq \frac{1}{f_{max}(\lambda_q)},$$

where $f_{max}(\lambda_q)$ is the maximal frequency of requests $\lambda_q$, and $\lambda_o$ is response to the request. Otherwise, the system will be not able to process requests in real time.

The system specification consists of a set of distributed algorithms (tasks). Our scheduling method is based on the assumption that the worst case is when all tasks start at the same time, which corresponds to the simultaneous appearance of all requests. Thus, all tasks are scheduled in a fixed order and are activated in certain time frames. When the system receives a new request, it will be processed during the next activation of the corresponding task. Therefore, time constraints should include this delay, i.e. the task should be scheduled with a period equal to $t_{max}(\lambda_q, \lambda_o) / 2$.

The goal of optimization is the minimization of the cost of outsourcing the network infrastructure to cloud providers. The method schedules tasks and transmissions based on available resources. Because in each step it should decide which resource to use to execute the next process or transmission, it should take into
consideration other preferences. In our method, the least utilized resource is chosen, which corresponds to a minimization of the peak workload of servers or communication links. In this way, we may avoid blocking the resource by our application and it may also be used for other applications.

3.2. Algorithm of synthesis

Synthesis is performed using the greedy algorithm, which schedules processes according to their priority. It starts from the allocated network infrastructure where all processes are assigned to resources with the highest performance. Next, while scheduling does not violate any time constraint, the method tries to reduce the cost of IaaS services by modifying the network infrastructure. The methodology presented repeats the following steps until all processes and transmissions are scheduled:

- Selects the next process, schedules it, and schedules all transmissions of messages sent to this process; next, the dynamic task graph is created.
- If the cost of IaaS services is not reduced, the most expensive resources are changed to cheaper ones; no time constraint should be violated; resources with the maximal performance are considered first.

The algorithm is constructive; thus, in each step, it should be able to verify whether, after scheduling the next task, it is still possible to obtain the valid system. For this purpose, a dynamic task graph (DTG) is created. All tasks are simultaneously analyzed according to their order of execution, assuming that processes and transmissions will be executed by the fastest resources. In the system specification, only the first message received by a process is relevant, and so all other messages are temporarily neglected. In this way, the specification is converted into a task graph. Next, the task graph is scheduled using the ‘as soon as possible’ (ASAP) method. All paths in the DTG that are embraced by time constraints are called critical paths. For each critical path $p_i$, the laxity is defined as follows:

$$L(p_i) = t_{\text{max}}(p_i) - t(p_i),$$

where $t_{\text{max}}(p_i)$ is a deadline specified for path $p_i$, and $t(p_i)$ is the time when the last task on the path finishes its execution. Each process has assigned priority in reverse order of the laxity of the path containing this process.

Processes are scheduled according to their priorities. After scheduling each task, the DTG is modified and verified against time requirements. If any deadline is overrun, then the network infrastructure is modified to enable the earlier schedule of the last task and the method proceeds with the next tasks.

Details of the algorithm are presented in Figure 5. First, $DTG_i$ for each task is created, and all processes are assigned to the best resources and scheduled using the ASAP method. Next, critical paths are computed, and if any path runs over the deadline, the algorithm stops (it is not possible to construct the system even using the fastest resources). Next, the initial list $L$ of processes is created and sorted according to an ascending order of priorities. The initial list contains only initiators. In the main loop of the algorithm, consecutive processes are removed from list $L$ and scheduled. After successful scheduling, the DTG graph is reconstructed, new priorities are computed, and all successors of the scheduled task are added to list $L$. The cost of IaaS computation services may be reduced by replacing the set of servers (nodes) with cheaper ones. The same result for IaaS communication services may be achieved by decreasing the reserved bandwidth of connections (assuming that lower bandwidth means a lower cost of the communication service). If it is not possible to find a feasible schedule after performing the above modifications, the set of resources $\Psi'$ required for finding the
schedule will be added to the set of allocated resources $\Pi_{IaaS}$, or resources will be replaced by faster ones. As a result, the required network infrastructure and the statically scheduled DTG are received. The total cost of IaaS services required by the application will also be given.

```java
/* Construction of DTG graphs for each task */
$\Pi$ = available network infrastructure;
Find the best performance of resources available in the database: $P_{p_{\text{cpu, best}}}$, $b_{c_{\text{best}}}$;
for each action $A_i$ do
  for each process $v_i$ in $A_i$ do assign $t(v_i) = w(v_i) / P_{p_{\text{cpu, best}}}$ to $v_i$;
  for each message $m_i$ in $A_i$ do assign $t(c_i) = l(m_i) / b_{c_{\text{best}}}$ to $m_i$;
for each action $A_i$ do
  Create temporary dynamic graph DTG;
ASAP scheduling and assigning priorities to each process $v_i$ in all DTGs;
for each DTG, do
  for each critical path $p_{i_k}$ in DTG, do
    if $t(p_{i_k}) > t_{\text{med}}(p_{i_k})$ then stop - it is not possible to find the solution;
  Create an ordered list of initiator processes starting from the lowest priority $L = \{v_{i_1},...,v_{i_k}\}$;
/* Scheduling */
while list $L \neq \emptyset$ do
  $v_i$ = first($L$); Remove $v_i$ from $L$;
  Find $\Psi = \{S, C, R\}$ in $\Pi$ such that $C_r(v_i)$ is minimal;
  Assign $v_i$ to $S_i$ and incoming messages of $v_i$ to $C_i$;
  Reschedule processes in each DTG;
for each DTG, do
  for each critical path $p_{i_k}$ in DTG, do
    if $t(p_{i_k}) > t_{\text{med}}(p_{i_k})$ then
      Find $\Psi' = \{S, C, R\}$ from $\Pi_{\text{new}}$ such that all time constraints are satisfied and $C(\Psi')$ is minimal;
      Assign $v_i$ to $n(S_i)$ and incoming messages of $v_i$ to $C_i$;
      Reschedule processes in each DTG;
Compute new priorities and update $L$;
```

Figure 5. Algorithm of synthesis.

The algorithm for minimizing the cost of IaaS services is presented in Figure 6. The system performance may be increased by adding a new server from set $\Pi$ or one node to previously allocated servers (for faster execution of task $v_{i_1}$), and/or by adding new connections/routers from $\Pi$ or by increasing the reserved bandwidth of allocated links (to finish transmissions incoming to task $v_{i_1}$ earlier). First, all possible single modifications are analyzed, and for each one the cost is computed. If there exists a modification for which a violation of the deadline will be compensated (i.e. $\Delta t \geq \Delta t_r$, where $\Delta t_r = t(p_{i_k}^k) - t_{\text{max}}(p_{i_k}^k)$), then the modification with minimal cost is returned. Otherwise, all combinations of single modifications are analyzed according to the ascending order of costs. The algorithm will finish when the first set of modifications compensating for the overrun deadline is found.

4. Example
As an example to demonstrate our methodology, we present the design of an adaptive navigation system for a smart city. We assume that all cars are equipped with GPS navigation devices (GDs) that are able to communicate with the Internet using wireless communication (we assume that the network of access points
Figure 6. Algorithm for cost minimization.

```
Cost_{mn} = 0;
\Delta t = t\{ p_i^k \} - t_{new}\{ p_i^k \};
\Pi_{host} = \text{set of allocated resources}
\Omega = \varphi; /* set of resources to add */
for each cl_i in \Pi_{host} do
    for each incoming transmission m_i do
        Temporarily add cl_i and router if required (or change bandwidth of cl_i) to \Pi and assign m_i to it;
        \Delta t = t\{ p_i^k \} - t\{ p_i^k \}; /* \Delta t is the time before temporarily adding resource to \Pi and t - the time after adding resource */
        if \Delta t > 0 then \Omega = \Omega \cup \{ p_i^k \}; \Pi = \Pi \cup \Omega;
    for each s_i and N_i in \Pi_{host} do
        Temporarily add s_i or N_i to \Pi and assign \nu_i to it;
        \Delta t = t\{ p_i^k \} - t\{ p_i^k \};
        if \Delta t > 0 then \Omega = \Omega \cup \{ s_i, N_i \}; \Pi = \Pi \cup \Omega
    if exists \nu_i \in \Omega such that \Delta t \geq \Delta t_i then
        return MinCost(\nu_i);
    for each subset \alpha = \{ \nu_1, \ldots, \nu_k \} from \Omega do
        Temporarily add \alpha to \Pi_{host} and assign \nu_i and incoming transmissions to it;
        \Delta t = t\{ p_i^k \} - t\{ p_i^k \};
        if \Delta t > 0 then return \alpha;
```

covers the whole city). The GDs send requests to the RTIoT system. Requests contain information about the current position, destination, and user preferences. The system then finds the optimal route and sends a response to the GD. Since the GD expects a response in a reasonable amount of time, the system should satisfy real-time constraints. We assume that the time in which the GD has to get an answer must be no longer than 5 s. The idea of such a system is based on the adaptability, i.e. the system may take into consideration traffic information and traffic impediments (e.g., car accidents), and it may construct different routes for the same destinations to avoid traffic jams.

Since the system may receive thousands of requests per second, the centralized system may not be able to handle all requests due to the communication bottleneck. Therefore, we propose the distributed system. The city is partitioned into sectors, and routes through each sector are computed by different processes (Figure 7). Each process also receives requests and sends responses from/to positions in the corresponding sector. Thus, the function of the system may be specified as a set of distributed algorithms, similar to the echo algorithm. In our example, the specification consists of 8 tasks, and in each task another process is the initiator. The initiator receives all requests coming from the corresponding sector, computes all possible routes to adjacent sectors, and sends the information about routes to adjacent processes. When messages reach the destination sector, the best route is selected and information about it is sent back to the initiator.

Assume that a cloud provider offers 9 servers and 4 bandwidths for communication services, and assume that the parameters of available resources are as follows. In Table 1, servers available in the cloud and costs of IaaS computing services are presented. Table 2 shows available bandwidths of communication links and the cost of IaaS communication services. Execution times of all processes for all available servers are presented in Table 3. Transmission times for available communications bandwidths are presented in Tables 4 and 5. A database of resources available in the cloud that may be allocated is presented in Figure 8. The time constraint \( t_{\text{max}} \) equals 5 s.
Table 1. Cost of available IaaS services for one server.

<table>
<thead>
<tr>
<th>Server</th>
<th>Processor</th>
<th>Per hour</th>
<th>Per month</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1.7 GHz</td>
<td>0.004 $</td>
<td>2.88 $</td>
</tr>
<tr>
<td>S2</td>
<td>2.4 GHz</td>
<td>0.008 $</td>
<td>5.76 $</td>
</tr>
<tr>
<td>S3</td>
<td>2 × 1.7 GHz</td>
<td>0.007 $</td>
<td>5.04 $</td>
</tr>
<tr>
<td>S4</td>
<td>2 × 2.4 GHz</td>
<td>0.014 $</td>
<td>10.08 $</td>
</tr>
<tr>
<td>S5</td>
<td>4 × 1.7 GHz</td>
<td>0.013 $</td>
<td>9.36 $</td>
</tr>
<tr>
<td>S6</td>
<td>4 × 2.4 GHz</td>
<td>0.025 $</td>
<td>18 $</td>
</tr>
</tbody>
</table>

Table 2. Cost of available IaaS communication services.

<table>
<thead>
<tr>
<th>Link</th>
<th>Bandwidth (Mbps)</th>
<th>Per hour</th>
<th>Per month</th>
</tr>
</thead>
<tbody>
<tr>
<td>L × 1</td>
<td>1</td>
<td>0.0001 $</td>
<td>0.072 $</td>
</tr>
<tr>
<td>L × 2</td>
<td>5</td>
<td>0.0010 $</td>
<td>0.72 $</td>
</tr>
<tr>
<td>L × 3</td>
<td>10</td>
<td>0.0028 $</td>
<td>2.04 $</td>
</tr>
<tr>
<td>L × 4</td>
<td>20</td>
<td>0.0069 $</td>
<td>5 $</td>
</tr>
</tbody>
</table>

Table 3. Execution times for available servers.

<table>
<thead>
<tr>
<th>Process</th>
<th>Workload</th>
<th>Server S1 T_i [ms]</th>
<th>T_i [ms]</th>
<th>Server S3 T_i [ms]</th>
<th>T_i [ms]</th>
<th>Server S4 T_i [ms]</th>
<th>T_i [ms]</th>
<th>Server S5 T_i [ms]</th>
<th>T_i [ms]</th>
<th>Server S6 T_i [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>920</td>
<td>541</td>
<td>383</td>
<td>541</td>
<td>383</td>
<td>541</td>
<td>383</td>
<td>541</td>
<td>383</td>
<td></td>
</tr>
<tr>
<td>G2</td>
<td>1020</td>
<td>600</td>
<td>425</td>
<td>600</td>
<td>425</td>
<td>600</td>
<td>425</td>
<td>600</td>
<td>425</td>
<td></td>
</tr>
<tr>
<td>G3</td>
<td>730</td>
<td>429</td>
<td>304</td>
<td>429</td>
<td>304</td>
<td>429</td>
<td>304</td>
<td>429</td>
<td>304</td>
<td></td>
</tr>
<tr>
<td>G4</td>
<td>990</td>
<td>582</td>
<td>412</td>
<td>582</td>
<td>412</td>
<td>582</td>
<td>412</td>
<td>582</td>
<td>412</td>
<td></td>
</tr>
<tr>
<td>G5</td>
<td>520</td>
<td>306</td>
<td>217</td>
<td>306</td>
<td>217</td>
<td>306</td>
<td>217</td>
<td>306</td>
<td>217</td>
<td></td>
</tr>
<tr>
<td>G6</td>
<td>900</td>
<td>529</td>
<td>375</td>
<td>529</td>
<td>375</td>
<td>529</td>
<td>375</td>
<td>529</td>
<td>375</td>
<td></td>
</tr>
<tr>
<td>G7</td>
<td>720</td>
<td>424</td>
<td>300</td>
<td>424</td>
<td>300</td>
<td>424</td>
<td>300</td>
<td>424</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>G8</td>
<td>860</td>
<td>506</td>
<td>358</td>
<td>506</td>
<td>358</td>
<td>506</td>
<td>358</td>
<td>506</td>
<td>358</td>
<td></td>
</tr>
</tbody>
</table>
The initial architecture is presented in Figure 9. It consists of the fastest resources corresponding to the most expensive services. The cost of IaaS services for this architecture is $0.0638/h. First, 8 task graphs are constructed (Figure 10). Each task graph specifies the sequence of processes activated by requests coming from different sectors of the city. All task graphs are scheduled using the initial architecture. The performance of the system is sufficient, and thus the next step is the minimization of the cost of IaaS services. The algorithm tries to replace the resources assigned to execute the following processes with cheaper ones. The following optimization steps are presented in Figure 11. Finally, the cost was reduced by about 40% to $0.037/h. The final architecture is presented in Figure 12. In Figure 13, the Gantt chart presenting the scheduling of all processes is shown. We may observe high utilization of all servers.

The frequency of task activation depends on the number of requests appearing during the given time period. For a large number of requests, the system will require more computing power. Thus, the cost of IaaS services strongly depends on the maximal estimated traffic in the city. Figure 14 presents the dependence between the number of requests and the cost of IaaS services. Therefore, it is very important to estimate the maximal intensity of traffic to make the system guarantee quality of service.

### Table 4. Times of outgoing transmissions for servers G1–G4.

<table>
<thead>
<tr>
<th>Lp.</th>
<th>From</th>
<th>To</th>
<th>Transmission size</th>
<th>L × 1</th>
<th>L × 2</th>
<th>L × 3</th>
<th>L × 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>[kb]</td>
<td>T_i</td>
<td>T_i</td>
<td>T_i</td>
<td>T_i</td>
</tr>
<tr>
<td>1</td>
<td>G1</td>
<td>G2</td>
<td>425</td>
<td>425</td>
<td>85</td>
<td>42.5</td>
<td>21.25</td>
</tr>
<tr>
<td>2</td>
<td>G1</td>
<td>G4</td>
<td>405</td>
<td>405</td>
<td>81</td>
<td>40.5</td>
<td>20.25</td>
</tr>
<tr>
<td>3</td>
<td>G2</td>
<td>G1</td>
<td>310</td>
<td>310</td>
<td>62</td>
<td>31</td>
<td>15.5</td>
</tr>
<tr>
<td>4</td>
<td>G2</td>
<td>G3</td>
<td>405</td>
<td>405</td>
<td>81</td>
<td>40.5</td>
<td>20.25</td>
</tr>
<tr>
<td>5</td>
<td>G2</td>
<td>G4</td>
<td>455</td>
<td>455</td>
<td>91</td>
<td>45.5</td>
<td>22.75</td>
</tr>
<tr>
<td>6</td>
<td>G3</td>
<td>G2</td>
<td>485</td>
<td>485</td>
<td>97</td>
<td>48.5</td>
<td>24.25</td>
</tr>
<tr>
<td>7</td>
<td>G3</td>
<td>G7</td>
<td>375</td>
<td>375</td>
<td>75</td>
<td>37.5</td>
<td>18.75</td>
</tr>
<tr>
<td>8</td>
<td>G4</td>
<td>G1</td>
<td>415</td>
<td>415</td>
<td>83</td>
<td>41.5</td>
<td>20.75</td>
</tr>
<tr>
<td>9</td>
<td>G4</td>
<td>G2</td>
<td>520</td>
<td>520</td>
<td>104</td>
<td>52</td>
<td>26</td>
</tr>
<tr>
<td>10</td>
<td>G4</td>
<td>G5</td>
<td>290</td>
<td>290</td>
<td>58</td>
<td>29</td>
<td>14.5</td>
</tr>
<tr>
<td>11</td>
<td>G4</td>
<td>G6</td>
<td>460</td>
<td>460</td>
<td>92</td>
<td>46</td>
<td>23</td>
</tr>
<tr>
<td>12</td>
<td>G4</td>
<td>G7</td>
<td>280</td>
<td>280</td>
<td>56</td>
<td>28</td>
<td>14</td>
</tr>
</tbody>
</table>

### Table 5. Times of outgoing transmissions for servers G5–G8.

<table>
<thead>
<tr>
<th>Lp.</th>
<th>From</th>
<th>To</th>
<th>Transmission size</th>
<th>L × 1</th>
<th>L × 2</th>
<th>L × 3</th>
<th>L × 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>[kb]</td>
<td>T_i</td>
<td>T_i</td>
<td>T_i</td>
<td>T_i</td>
</tr>
<tr>
<td>13</td>
<td>G5</td>
<td>G4</td>
<td>300</td>
<td>300</td>
<td>60</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>G5</td>
<td>G6</td>
<td>430</td>
<td>430</td>
<td>86</td>
<td>43</td>
<td>21.5</td>
</tr>
<tr>
<td>15</td>
<td>G6</td>
<td>G4</td>
<td>490</td>
<td>490</td>
<td>98</td>
<td>49</td>
<td>24.5</td>
</tr>
<tr>
<td>16</td>
<td>G6</td>
<td>G5</td>
<td>405</td>
<td>405</td>
<td>81</td>
<td>40.5</td>
<td>20.25</td>
</tr>
<tr>
<td>17</td>
<td>G6</td>
<td>G7</td>
<td>300</td>
<td>300</td>
<td>60</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>18</td>
<td>G6</td>
<td>G8</td>
<td>360</td>
<td>360</td>
<td>72</td>
<td>36</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>G7</td>
<td>G3</td>
<td>325</td>
<td>325</td>
<td>65</td>
<td>32.5</td>
<td>16.25</td>
</tr>
<tr>
<td>20</td>
<td>G7</td>
<td>G4</td>
<td>305</td>
<td>305</td>
<td>61</td>
<td>30.5</td>
<td>15.25</td>
</tr>
<tr>
<td>21</td>
<td>G7</td>
<td>G6</td>
<td>380</td>
<td>380</td>
<td>76</td>
<td>38</td>
<td>19</td>
</tr>
<tr>
<td>22</td>
<td>G7</td>
<td>G8</td>
<td>500</td>
<td>500</td>
<td>100</td>
<td>50</td>
<td>25</td>
</tr>
<tr>
<td>23</td>
<td>G8</td>
<td>G6</td>
<td>400</td>
<td>400</td>
<td>80</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>24</td>
<td>G8</td>
<td>G7</td>
<td>500</td>
<td>500</td>
<td>100</td>
<td>50</td>
<td>25</td>
</tr>
</tbody>
</table>
Figure 8. Database of resources in cloud.

Figure 9. Initial network infrastructure.
5. Conclusions
In this paper, the methodology for the synthesis of reactive, real-time cloud applications accordant with the IoT concept was presented. We developed the architectural model of the reactive RTIoT system and we proposed the method of specification for such systems in the form of a set of distributed algorithms. Next, the method of synthesis that guarantees the fulfillment of all time requirements was proposed. The method schedules all processes and transmissions on network resources supported by cloud providers, while the cost of IaaS services is minimized. Finally, we presented the design process of the sample RTIoT system, which underlines the
advantages of our methodology. To the best of our knowledge, it is the first methodology of synthesis for real-time IoT systems.

Figure 12. Target architecture.

Figure 13. Gantt chart for target scheduling of processes.

Figure 14. Dependence between the number of requests and the cost of IaaS services.
In our approach, we use a heuristic greedy algorithm for scheduling and allocation of new resources. In future work, we will consider developing a more sophisticated method of optimization as well as more advanced methods for the worst-case analysis. Reactive RTIoT systems are a new challenge for future IoT systems. We think that, in the future, RTIoT systems will constitute an important class of IoT systems, and thus efficient design methods will be very desirable.

References