Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 743–752
DOI: 10.15439/2014F234
ACSIS, Vol. 2

Synthesis of Real Time Distributed Applications for Cloud Computing

Stanisław Deniziak
Cracow University of Technology, Department of Computer Engineering
Warszawska 24, 31-155 Cracow, Poland
Kielce University of Technology, Department of Computer Science
Al. Tysiaclecia Panstwa Polskiego 7, 25-314 Kielce, Poland, Email: sdeniziak@pk.edu.pl

Sławomir Bąk
Cracow University of Technology, Department of Computer Engineering
Warszawska 24, 31-155 Cracow, Poland
Email: sbak@pk.edu.pl

***Abstract*—This paper presents the methodology for the synthesis of real-time applications for the Infrastructure as a Service (IaaS) model of cloud computing. We assume that the function of the application is specified as a set of distributed echo algorithms with real-time constraints. Then our methodology schedules all tasks on available cloud infrastructure minimizing the total costs of the IaaS services, while satisfying all real-time requirements. It takes into account limited bandwidth of communication channels as well as the limited computation power of server nodes. The optimization is based on the iterative improvement algorithm, which has the capability of escaping from the local extrema, giving better results than greedy algorithms. The method starts from the fastest solution and in the next steps modifies the solution to reduce the cost of hiring the cloud infrastructure. We also present a sample application, that shows the benefits of using our methodology.**

***Index Terms*—cloud computing, Infrastructure as a Service, real-time system, distributed systems, system synthesis.**

I. Introduction

CLOUD computing recently has received significant attention as a new computing infrastructure. A cloud environment often has hundreds of thousands of processors with numerous disks interconnected by dedicated high-speed networks. There are three deployment models of cloud computing [1]. The first is the private cloud, it works specially for organization with private security and exclusive network. The second is the public cloud, it gives the maximum efficiency level in shared resources and it is protected by the cloud service provider. The third is the hybrid cloud, it combines the private and public. Cloud computing supports three types of services [2]:

- IaaS (Infrastructure as a Service) offers end users direct access to processing, storage, and other computing resources. IaaS allows users to configure resources, to run operating systems and to run application software on them. Examples of IaaS are: Amazon Elastic Compute Cloud (EC2), Rackspace and IBM Computing on Demand,
- PaaS (Platform as a Service) offers an operating system as well as suites of programming languages and software development tools that customers can use to develop their own applications. Examples of PaaS are: Microsoft Windows Azure and Google App Engine. PaaS gives end users control over application design, but does not give them control over the physical infrastructure,
- SaaS (Software as a Service) offers final applications that end users can access through a thin client (web browser). Examples of SaaS are: Gmail, Google Docs. The end users (customers) do not exercise any control over the design of the application, servers, networking and storage infrastructure.

Cloud computing is really changing the way, how and where the computing is going to be performed. 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 each product may be identified and traced in the Internet using RFID (Radio Frequency IDentification), NFC (Near Field Communication) or other wireless communication methods. Smart device not only incorporates 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 jam. In such case cloud applications are used to process requests sent by smart devices implementing client applications. Usually responses to the device should be sent during the limited time period. Therefore, this class of application is a real-time system.

Distributed Internet application requires 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 by using the Infrastructure as a Service (IaaS) model [3] of the cloud computing services [4]. IaaS together with a real-time cloud environment [5] seems the ideal platform for many real-time cloud applications. But to guarantee the quality of service

978-83-60810-58-3/$25.00

and minimize the cost of the system, efficient methods of mapping real-time applications onto IaaS should be developed.

Some studies [6], [7] consider resource allocation for cloud applications. The common focus of these works is the optimization of resource allocation from IaaS in respect of the cost. One of the previous method selecting resources from a cloud is based on the conception of the game theory [8]. The method optimizes the cost and the performance. This conception reflects the common characteristics of the physical position and bandwidth available between job and resources, and emphasizes on establishing a scheduling relationship between near entities. In resource scheduling, a choice of near and low-cost resources is a key criterion. Paper [9] also describes the scheduling algorithm for cloud computing. In this cost-based method, the set of computing resources with the lowest price are assigned to the user, according to the current supplier resource availability and a price. Another method, proposes scheduling of resources, based on genetic algorithm [10]. In this method, scheduling scheme is coded using integer sequence and a fitness function is based on influence degree. The genetic operations include selection, crossover, mutation and elitist selection. None of the above methods consider real-time requirements.

The use of the cloud infrastructure for real time computing is a quite new concept. Current work concerning Real Time Cloud Computing mainly concentrates on 2 domains: adopting existing web technologies to this new paradigm and developing software architectures for real-time applications. Recent studies [11]–[14] have been performed on the allocation of resources for real time tasks. Aymerich *et al.* [11] developed an infrastructure for a real-time financial system based on cloud computing technologies. Liu *et al.* [12] showed how to schedule real-time tasks with different utility functions. The real-time tasks are scheduled non-preemptively with the objective to maximize the total utility by using time utility function (TUF). Tsai *et al.* [13] discuss about a real-time database partitioning on cloud infrastructures. Kim *et al.* [14] investigate power-aware provisioning of resources for real-time cloud services. In their work the real-time constraint is specified in a Service Level Agreement (SLA) between customers and cloud providers. SLAs specify the negotiated agreements, including Quality of Service (QoS), such as deadlines. In such cloud models the service provider is responsible for the allocation resources. Their work examines power management while allocation of resources should meet the SLA. None of these studies consider a cost-efficient selection, from a set of different types of resources available in clouds, for real-time tasks.

The closest work to ours is that of Kumar *et al.* [15]. They develop an algorithm of resource allocation for applications with real-time tasks. They propose an EDF-greedy scheme and a scheme considers temporal overlapping to allocate resources efficiently. Unfortunately an EDF-greedy strategy may not give the lowest total cost, because of their tendency to be trapped in local minima of the cost.

In our work, we consider the IaaS model of the real-time cloud computing, where the user pays the cost of using the resources supported by the service provider. We present the methodology for the synthesis of reactive, real-time cloud applications specified as a set of distributed echo algorithms. The goal of our methodology is to find the distributed architecture of the application which will satisfy all user requirements. We developed an iterative improvement algorithm, which is able to escape from the local extrema, giving much better results than constructive algorithms. Presented method also minimizes the cost of IaaS services required for running the real-time application in the cloud environment.

The next section presents our assumptions and it defines the concept of real-time cloud computing used in our methodology. In section 3 the method of synthesis will be described. Section 4 presents example and experimental results demonstrating the advantages of the methodology. The paper ends with conclusions.

II. Problem statement

System synthesis is a process of automatic generation of the system architecture, starting from the formal specification of functional and non-functional requirements. 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, cost requirements define the maximal cost of the system, etc.

Functions of distributed systems are usually specified as a set of communicating tasks or processes. Since we consider real-time systems, hence time constraints are the main set of requirements. The model of the system specification used in our methodology will be described in p.1.

We use existing network infrastructure, hired from a cloud (IaaS), consisting of servers, routers and connections. If the current architecture does not guarantee that all time requirements will be met, 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 will be described in p.2, while requirements that are used in our methodology will be presented in p.3.

1. Functional specification

We assume that a real-time cloud application will process requests received from clients. The system should be able to process all requests during the required time period, i.e., for each real-time request a response should be sent before the specified deadline. We consider soft real-time processing [16], ensuring that the process will be completed at a given time depending on the constraints of quality of service. In

case of a large number of requests and a long time of processing, real-time processing will be possible only if massive parallel computing will be 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: system should be specified as a set of parallel processes using message passing communication,
(2) parallel request handling: huge number of requests may cause the 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 required set of events will appear. As a result, the task may generate other events. External input events will be called requests (Q), external output events are 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

$$\Lambda = Q \cup O \cup M = \{\lambda_i : i = 1, \ldots, r\} \quad (1)$$

For each event λ_i communication workload $\omega(\lambda_i)$ is defined. System activity is defined as the following function:

$$\Phi : C \times V \rightarrow \omega \times 2^{\Lambda} \quad (2)$$

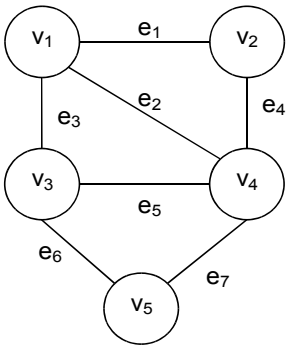
where C is an event expression (logical expression consisting of logical operators and Boolean variables representing events) and ω is the workload of the activated process.

Using function Φ it is possible to specify various classes of distributed algorithms. Fig. 1 presents sample echo algorithm [18] consisting of 5 processes. The algorithm consists of 10 actions. Each action is activated only once, when the corresponding condition will be equal to true. All actions except A_1 and A_6 contain alternative sub-actions. Only the first action, for which the condition will be satisfied, will be activated. According to the echo algorithm specification, process v_1 is the initiator, messages $m1_1$, ..., $m14_7$ are explorer messages, while $m15_1$, ..., $m25_7$ are echo messages (indices are added only for readability, mx_i means that message mx is associated with edge e_i in the graph, for the same reason, edge names in the event expressions mean any received message corresponding to this edge, e.g., $e_1 = m1_1 \mid m4_1 \mid m15_1$, $e_2 = m2_2 \mid m9_2 \mid m20_2$, etc.). Events x_1, ..., x_{11} are internal events, used for storing the state of processes between successive executions.

Since different requests may be processed by distinct algorithms, the function of a system may be specified using a set of functions Φ 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 consists of the finite number of actions and infinite loops are not allowed.

2. Real Time IaaS Architecture

The proposed architecture of RTCCI (Real Time Cloud Computing Infrastructure) is composed of two layers (Fig. 2): Network Layer (*NL*) and Server Layer (*SL*).



A_1: $\Phi(v_1,\{q_1\}) \rightarrow (5, \{m1_1,m2_2,m3_3\})$

A_2: $\Phi(v_2,\{m1_1\}) \rightarrow (4,\{x_1,m5_4\}) \mid \Phi(v_2,\{m10_4\}) \rightarrow (4,\{x_2,m4_1\})$

A_3: $\Phi(v_3,\{m3_3\}) \rightarrow (7,\{x_3, m7_5,m8_6\}) \mid \Phi(v_3,\{m11_5\}) \rightarrow (7,\{x_4,m6_3,m8_6\})$
$\mid \Phi(v_3,\{m13_6\}) \rightarrow (7,\{x_5,m6_3,m7_5\})$

A_4: $\Phi(v_4,\{m2_2\}) \rightarrow (6,\{x_6,m10_4,m11_5,m12_7\})\mid$
$\Phi(v_4,\{m5_4\}) \rightarrow (6,\{x_7,m9_2,m11_5,m12_7\})\mid$
$\Phi(v_4,\{m7_5\}) \rightarrow (6,\{x_8,m9_2,m10_4,m12_7\})\mid$
$\Phi(v_4,\{m14_7\}) \rightarrow (6,\{x_9,m9_2,m10_4,m11_5\})$

A_5: $\Phi(v_5,\{m8_6\}) \rightarrow (5,\{x_{10},m14_7\}) \mid \Phi(v_5,\{m12_7\}) \rightarrow (5,\{x_{11},m13_6,\})$

A_6: $\Phi(v_1,\{e_1\&e_2\&e_3\}) \rightarrow (10, \{r_1\})$

A_7: $\Phi(v_2,\{x_1\&e_1\&e_4\}) \rightarrow (4,\{m15_1\}) \mid \Phi(v_2,\{x_2\&e_1\&e_4\}) \rightarrow (4,\{m16_4\})$

A_8: $\Phi(v_3,\{x_3\&e_3\&e_5\&e_6\}) \rightarrow (3,\{m17_3\}) \mid \Phi(v_3,\{x_4\&e_3\&e_5\&e_6\}) \rightarrow (3,\{m18_5\}) \mid \Phi(v_3,\{x_5\&e_3\&e_5\&e_6\}) \rightarrow (3,\{m19_6\})$

A_9: $\Phi(v_4,\{x_6\&e_2\&e_4\&e_5\&e_7\}) \rightarrow (5,\{m20_2\}) \mid \Phi(v_4,\{x_7\&e_2\&e_4\&e_5\&e_7\}) \rightarrow (5,\{m21_4\}) \mid \Phi(v_4,\{x_8\&e_2\&e_4\&e_5\&e_7\}) \rightarrow (5,\{m22_5\}) \mid$
$\Phi(v_4,\{x_9\&e_2\&e_4\&e_5\&e_7\}) \rightarrow (5,\{m23_7\})$

A_{10}: $\Phi(v_5,\{x_{10}\&e_6\&e_7\}) \rightarrow (2,\{m24_6\}) \mid \Phi(v_5,\{x_{11}\&e_6\&e_7\}) \rightarrow (2,\{m25_7\})$

Fig 1. Sample specification of the echo algorithm

Layer *NL* consists of Communication Channels (*CC*) composed of routers and communication links. For each CL_i the available bandwidth $B(CC_i)$ is defined.

Layer *SL* contains servers (*S*) consisting of computational nodes N_i. Each N_i is characterized by performance P_i reserved for RTCC system, and it may be equipped with a network interface. Thus, each computational node may be connected to another communication link.

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 channels, that are used in the target architecture $\Pi_T = \{S, CC\}$ of the system, will be outsourced to the cloud provider.

The method starts from the initial architecture $\Pi_I = \{S', CC'\}$ consisting of the fastest resources. Next, the architecture is optimized by performing some modifications of Π_I, only resources supported by cloud providers are considered here. Our methodology minimizes the cost of hiring the network infrastructure by achieving the maximal utilization of all resources and by allocating 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 $L = \{CC'', S''\}$.

Communication channels $cc_i \in CC''$ are characterized by the maximal available bandwidth $B(cc_i)$, bandwidth $B_r(cc_i)$ reserved for the application and the price of communication service $Cr(cc_i)$ for each available bandwidth. Communication channel connects any pair of network interface ports. Thus, the time of transmission of packet D_i through communication channel cc_j is the following:

$$T(D_i) = \frac{l(D_i)}{B_r(cc_j)} \tag{3}$$

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

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

– n_S - the number of nodes, hence server s_i may be represented as a set $\{N_1, ..., N_{ns}\}$ of nodes,

– $Cr(s_i)$ - the cost of the computing services, the cost depends on the number of nodes allocated to the application, usually the cost function is not linear.

– $\{P_1, ..., P_{ns}\}$ - performance of each node.

The time required for executing process τ_i by the node N_j equals:

$$T(\tau_i) = \frac{w(\tau_i)}{P_j} \tag{4}$$

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

Fig. 2 presents a sample target architecture of RTCCI.

3. Requirements and constraints.

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

$$A_1 : \Phi(v_i, \lambda_x) \rightarrow \{\omega_1, \{\lambda_1\}\},\ A_s : \Phi(v_j, \lambda_s) \rightarrow \{\omega_s, \{\lambda_y\}\},\ \underset{1<k<s-1}{\forall} A_k \rightarrow A_{k+1} \tag{5}$$

where v_i, v_j are any processes and $A_k \rightarrow A_{k+1}$ means that action A_k generates events activating action A_{k+1}. Then, the time of execution of the given sequence of actions is defined as a sum of the execution times of all processes and a time of inter-process communication:

$$t(p(\lambda_x, \lambda_y)) = \sum_{i=1}^{s} \frac{\omega(A_i)}{P(A_i)} + \sum_{i=1}^{s} \frac{\omega(m_i)}{B_r(m_i)} \tag{6}$$

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, $B_r(m_i)$ is the reserved bandwidth of the channel used for sending the message. 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 request and receiving the response. Since the request may activate different sequences of actions until the response will be obtained, therefore the time constraint (deadline) is defined as:

$$t_{max}(\lambda_x, \lambda_y) = \underset{i}{MAX}(t(p_i(\lambda_x, \lambda_y))) \tag{7}$$

During the synthesis, processes and transmissions are scheduled and assigned to network resources. The method first assigns processes and transmissions to the fastest re-

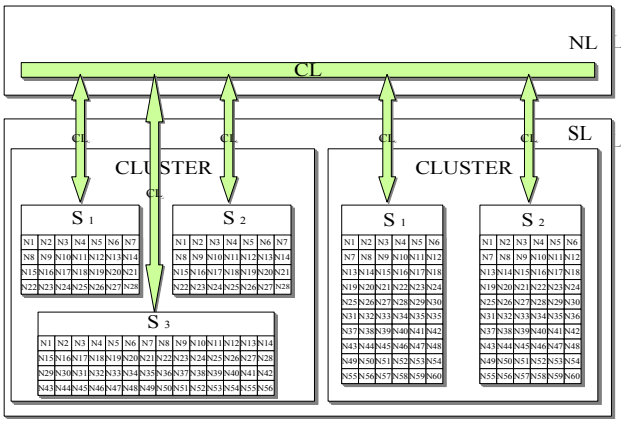


Fig 2. Sample target architecture

sources supported by cloud providers. This step verifies if it is possible to find the network infrastructure which fulfills all time constraints. Next, the cost is minimized by performing the following modifications:

- change communication channel to cheaper one, decreasing bandwidth $B_r(cc)$, for any allocated communication channel, in this way the cost of communication service $Cr(cc)$ may be reduced,
- change server s to cheaper one or reduce the number of allocated nodes, in this way the cost of computing services $Cr(s)$ 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 will cause violation of time requirements. Hence, the total cost of the IaaS service will be the following:

$$C_M = \sum_i Cr(cc_i) + \sum_j Cr(s_j) \tag{8}$$

The goal of our methodology is to minimize C_M.

III. Synthesis

Our method of synthesis starts from the formal specification of the system (as described in p. II.1) and tries to produce the optimal target architecture of the system, that satisfies all constraints. The method minimizes the cost of outsourcing the network infrastructure to the IaaS cloud provider.

1. Assumptions

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

$$t_{max}(\lambda_q, \lambda_o) \leq \frac{1}{f_{max}(\lambda_q)} \tag{9}$$

where $f_{max}(\lambda_q)$ is the maximal frequency of requests λ_q, λ_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 will start at the same time, this 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 will receive new request, it will be processed during the next activation of the corresponding task. Therefore, time constraints should include this delay, i.e., task should be scheduled with period equal:

$$\frac{t_{max}(\lambda_q, \lambda_o)}{2} \tag{10}$$

The goal of optimization is the minimization of the cost of outsourcing the network infrastructure to cloud providers. The method schedules tasks and transmissions on available cloud resources. We use an efficient iterative algorithm for finding the (sub-)optimal solution.

2. Dynamic task graph

The algorithm should be able to verify if after scheduling next task, it is still possible to obtain the valid system. For this purpose the 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. Since in the system specification only the first message received by a process is relevant, all other messages are temporarily neglected. In this way the specification is converted into task graph. Next, the task graph is scheduled using ASAP (As Soon As Possible) method.

3. Algorithm of the synthesis

In our earlier works [19], [20] synthesis is performed using the greedy algorithm, that schedules processes according to their priority. However, it is constructive algorithm and the obtained results are far from optimal, because the method is prone to be trapped in the local minima of the cost. In this paper, we present the iterative improvement algorithm, which is able to escape from the local minima, giving better results than constructive algorithm. An outline of the algorithm is shown on Fig. 3.

Gain is the difference of quality of the new solution and the current one. The quality of the solution is determined on the basis of several features of the target architecture. In our case, the quality of solutions is based on the cost of the system that satisfies all time constraints, i.e., the optimum is the cheapest system that meets the time constraints.

The algorithm starts from the initial architecture where all processes are assigned to resources with the highest performance, according to the rule: the biggest task to the fastest processor. For transmissions, also communication services with the highest bandwidth are reserved. Next, while any time constraint is not violated, the method tries to reduce the cost of IaaS services by modifying the network infrastructure using iterative improvement methods (refinement of the current result). The methodology repeats the following steps:

- remove the node or replace the resource with a cheaper one,

```
Generate initial solution Π^CUR
do{
  Π^BEST = Π^CUR;
  gain = 0;
  while((Π'=refine(Π^BEST)) != Φ){
   gain = Q(Π') - Q(Π^CUR);
   if(gain > 0)
     Π^CUR = Π';
  }
}while(gain > 0);
```

Fig 3. An outline of the iterative improvement algorithm.

- create a dynamic task graph,
- schedule all processes and transmissions.

The solution giving the best gain is chosen to the next step. The algorithm terminates when there is no solution that can improve the total cost of the system.

The quality of the solution determines the gain of the improvement. The aim of the algorithm is to minimize the cost, thus the main system feature determining the gain should be the cost of the system. However, driving refinement only according to the optimisation goal usually leads to trapping the algorithm in local minima (the greedy approach). Hence it is appropriate to define the quality of the solution using also other features of the solution. It should inhibit the greed of the algorithm. For this purpose laxity L is introduced. The laxity is defined as follows:

$$L = t_{max} - t_{cur}, L \geq 0 \tag{11}$$

where t_{cur} is the execution time for the current solution.

At each step, various modifications of the current system are considered. Each modification may change the cost and/or the latency of the solution. Quality (Q) of the modified system is defined as follows:

$$Q = \begin{cases} \frac{C_{BEST} \cdot L_{CUR}}{C_{CUR} \cdot L_{BEST}}, when\ \Delta C < 0 \text{ and } L_{CUR} \geq 0 \\ 0, when\ \Delta C > 0 \text{ or } L_{CUR} < 0 \end{cases} \tag{12}$$

where:

$$\Delta C = C_{CUR} - C_{BEST} \tag{13}$$

L_{CUR} and C_{CUR} are the features of the current result, L_{BEST} and C_{BEST} are the features of the best result, found in the previous iterations.

The quality is defined as the ratio of the previous cost to the cost after modification. If the latency is also changed, then the quality is modified by the percentage of the latency increase. If there is no reduction of the system cost, then the quiality equals 0, i.e., modification will be rejected. This condition guarantees that the algorithm is convergent. The quality is also equal to 0 when a time constraint is violated, i.e., $L_{CUR} < 0$.

Solutions that do not lead to a gain greater than 0 are rejected. The quality prefers solutions with the greatest reductions of cost and greater increase of the performance of the system. If all modifications do not reduce the cost of the system ($\Delta C=0$) then the solution with the greatest increase of the system performance is taken as the best to the next step.

At each step of the algorithm, various modifications of the current solution are considered and solution that gives the highest quality is chosen to the next step. Since the quality Q depends also on the increase of laxity, therefore the greed of the algorithm will be reduced, i.e., instead of the modification reducing cost the algorithm may select modification that more reduces the laxity. Higher laxity means more possiblilities of improvements in the next steps.

In order to minimize the cost of the system, in the algorithm the following modifications are considered:

(1) Change the node from the cloud to cheaper one and move tasks to it.
(2) Replace the communication link to cheaper one.
(3) Remove one node and move all assigned tasks to other nodes.

In the case where more than one task is allocated to the resource, it is necessary to schedule tasks. The FIFO scheduling method is used for this purpose. The refinement process is presented on Fig. 4. It consists of 3 loops, each loop evaluates all possible modifications of the system architecture. Only systems with quality greater than 0 are taken into consideration. We assume that the process *refine* returns the architecture, then after next activation it continues its execution. The process terminates after analyzing all possible modifications of the initial architecture. Architecture with the highest quality is taken as an input to the next step of the algorithm.

IV. Example

As an example demonstrating our methodology we present the design of an adaptive navigation system for a smart city [19]. We assume that all cars are equipped with GPS navigation devices (GD), that are able to communicate with the Internet using wireless communication (we assume that the network of access points covers the whole city). GD devices send requests to RTCC system. Requests contain information about current position, the destination and user preferences. Then, the system finds the optimal route and sends response to GD device. Since GD expects the response in a reasonable time, then the system should satisfy real-time constraints. We assume that the time in which the GD device has to get an answer must be no longer than 5 seconds. The idea of such system is based on the adaptability, i.e., the system may take into consideration traffic information, 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, routes through each sector are computed by different processes. 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 6 to 12 tasks, 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 will reach the destination sector, then the best route is selected and information about it is sent back to the initiator.

```
refine(Π^CUR){
    for each X_i ∈ Π^CUR do {
      Π′ = Π^CUR − X_i;
      for all X_j in IaaS do
         if C(X_j ) < C(X_i) then {
           Π″ = Π′ + X_j;
           for each v_k ∈ X_i do  //transfer tasks from X_i to X_j
             Assign v_k to X_j;
           if Q(Π″) > 0 then
              return Π″;
         }
    }
    for each CL_i ∈ Π^CUR do {
      Π′ = Π^CUR − CL_i;
      for all CL_j in IaaS do
        if C(CL_j ) < C(CL_i) then {
           Π″ = Π′ + CL_j;
          for(cl_k ∈ CL_i){  //transfer transmission from CL_i to CL_j
             Assign cl_k to Cl_j;
           if Q(Π″) > 0 then
              return Π″;
        }
    }
    for each X_i ∈ Π_CUR do {
     Π″ = Π^CUR − X_i;
     for each v_k ∈ X_i do {  //transfer tasks from X_i to other resource from Π^CUR
       Find resource X_j ∈ Π" such, that L(Π")is maximal after
         assigning v_k to X_j;
       Assign v_k to X_j;
     }
     if Q(Π″) > 0 then
       return Π″;
    }
   return Φ;
}
```

Fig 4. Synthesis algorithm for cost minimization.

Assume that a cloud provider offers 13 servers and 4 bandwidths for communication services (Fig. 5), and assume that parameters of available resources are known. In Table I available bandwidths of communication links and the cost of IaaS communication services are presented. Table II shows parameters of servers available in the cloud and costs of IaaS computing services are presented. The time constraint t_{max} equals 5 s.

Some dynamic tasks graph constructed for the best solution are presented on Fig 6. On Fig 7 the Gantt chart presenting the scheduling of all processes is presented. We may observe high utilization of all servers.

TABLE I COST OF AVAILABLE IAAS COMMUNICATION SERVICES

Link	Bandwidth (Mbps)	Per hour
Lx1	1	0.0001$
Lx2	5	0.0010$
Lx3	10	0.0028$
Lx4	20	0.0069$

TABLE II: COST OF AVAILABLE IAAS SERVICES FOR ONE SERVER.

Server	Processor	Per hour
S1	1.7 GHz	0.004 $
S2	2.4 GHZ	0.008 $
S3	2 × 1.7 GHz	0.007 $
S4	2 × 2.4 GHz	0.014 $
S5	4 × 1.7 GHz	0.013 $
S6	4 × 2.4 GHz	0.025 $
S7	4 x 1.5 GHz	0.012 $
S8	4 x 1.2 GHz	0.01 $

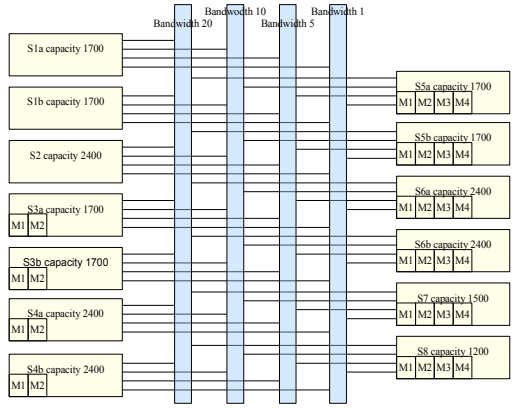


Fig 5. Database of resources available in the cloud.

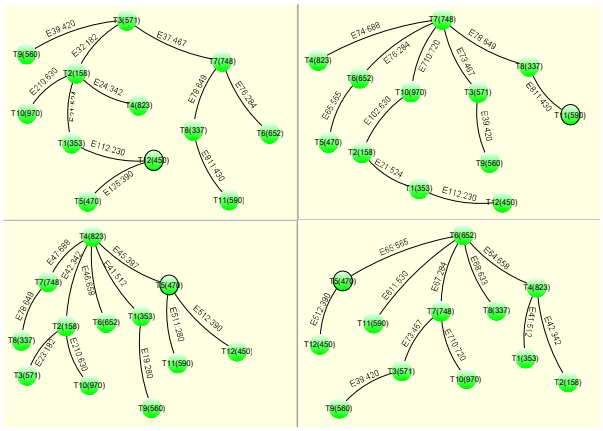


Fig 6. Several sample tasks graph

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. Fig. 8 and Table III present the dependence between the number of requests and the cost of IaaS services for greedy and iterative improvement algorithm. To allocate resources from the IaaS, iterative improvement algorithm produced much better results than greedy algorithm. The comparison is shown in Table III. Our algorithm allows the end users of IaaS to reduce the cost of hiring cloud resources by over 50%.

V. CONCLUSIONS

We analyze the problem of allocating resources for real-time tasks such that the cost is minimized and all the deadlines are met. In this paper the methodology for the synthesis of reactive, real-time cloud applications accordant with the Cloud Computing concept, was presented. We developed the architectural model of the reactive RTCC system and we proposed the method of specification for such systems, in the form of a set of distributed Echo 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 RTCC system, which underlines the advantages of our methodology above greedy algorithm.

In our approach we use iterative improvement algorithm for scheduling and allocation of new resources and we show its advantage over the heuristic greedy algorithm. In the future work we will consider developing a more sophisticated method of optimization as well as more advanced methods for the worst case analysis. Reactive RTCC systems are a new challenge for future Cloud Computing. We believe that in the future, RTCC systems will constitute an important class of Cloud Computing systems, thus efficient design methods will be very desirable.

REFERENCES

[1] IBMSmartCloud http://www.ibm.com/cloud-computing/us/en/what-is-cloud-computing.html. Last access, April 2014

[2] C. S. Yoo, “Cloud Computing: Architectural and Policy Implications”. *Review of Industrial Organization*, June 2011, 38.4: 405-421, http://dx.doi.org/10.1007/s11151-011-9295-7.

[3] A. Amies, H. Sluiman, QG. Tong and GN Liu, “Infrastructure as a Service Cloud Concepts. Developing and Hosting Applications on the Cloud” IBM Press, 2012.

[4] R. Buyya, J. Broberg, A. Goscinski, “Cloud Computing: Principles and Paradigms" New York, USA: Wiley Press., 2011. pp. 1–44, http://dx.doi.org/10.1002/9780470940105.

[5] D. Kyriazis et al, “A Real-time Service Oriented Infrastructure” *Annual International Conference on Real-Time and Embedded Systems (RTES 2010)*. November 2010, Singapore. pp. 39-44, http://dx.doi.org/10.5176/978-981-08-7654-8_R-47.

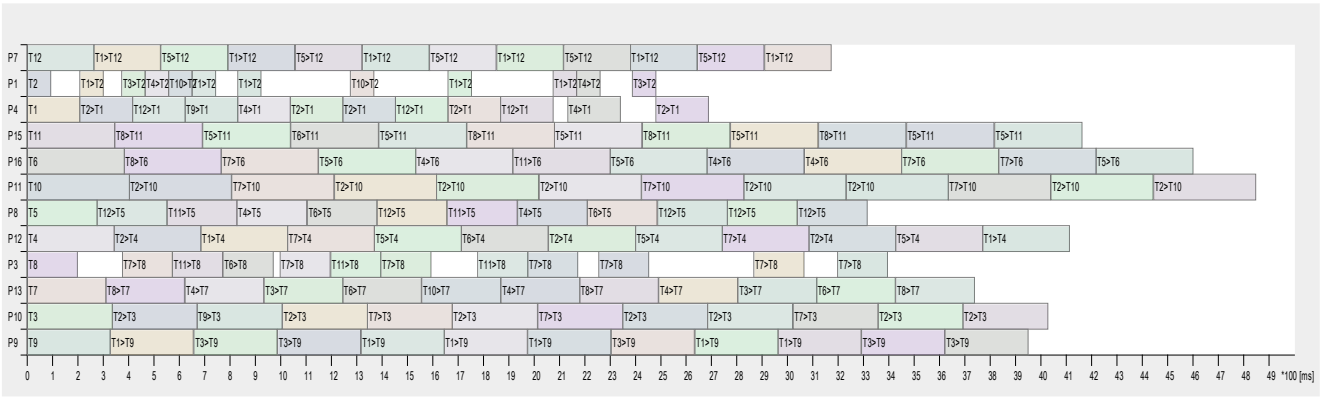


Fig 7. Gantt chart for target scheduling of processes

TABLE III TIME AND COST OF TASKS FOR GREEDY AND ITERATIVE IMPROVEMENT ALGORITHM

Lp.	Number of tasks	Greedy algorithm		Iterative improvement algorithm	
		Time [ms]	Cost [$/h]	Time [ms]	Cost [$/h]
1	36	4658.3333	0.01600	4658.3333	0.01600
2	49	4800.8333	0.02200	4884.3333	0.02155
3	64	4986.6667	0.05035	4346.6667	0.02480
4	81	4946.0000	0.05600	4938.0000	0.03010
5	100	4872.8265	0.10625	4921.0000	0.04550
6	121	4868.2500	0.13320	4714.2235	0.05655
7	144	4907.0000	0.13945	4987.0000	0.08075
AVERAGE COST		0,07475		0,03951	

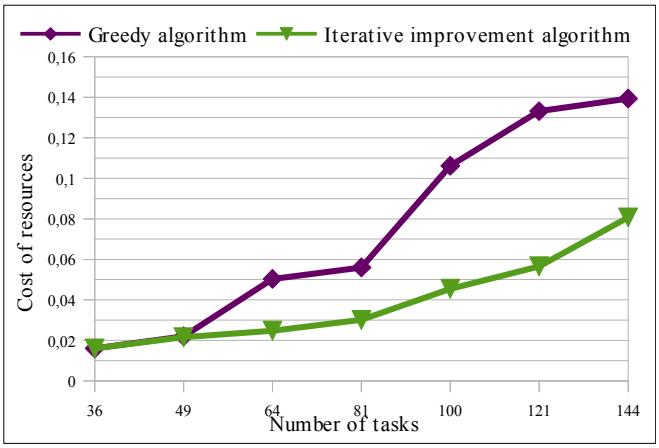


Fig 8. Dependence between the number of requests and the cost of IaaS services for greedy and iterative improvement algorithm.

[6] R. Huang, H. Casanova, A. A. Chien, "Automatic resource specification generation for resource selection" *ACM/IEEE Conference on Supercomputing*, November 2007, Reno, pp 1–11, http://dx.doi.org/10.1145/1362622.1362638.

[7] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good, "The cost of doing science on the cloud: the montage example" *ACM/IEEE Conference on High Performance Computing, Networking, Storage and Analysis*, November 2008, Austin, pp 1–12, http://dx.doi.org/10.1109/SC.2008.5217932.

[8] L. Mengkun, C. Ming, X. Jun, "Cloud Computing: A Synthesis Models for Resource Service Management" *2010 Second International Conference on Communication Systems, Networks and Applications (ICCSNA 2010),* vol.2, June 2010, Hong Kong, pp. 208-211, http://dx.doi.org/10.1109/ICCSNA.2010.5588886.

[9] Y. Zhi, Y. Changqin, L. Yan, "A Cost-based Resource Scheduling Paradigm in Cloud Computing" *12th International Conference on Parallel and Distributed Computing, Applications and Technologies*, October 2011, Washington, pp. 417-422, http://dx.doi.org/10.1109/PDCAT.2011.1.

[10] W. Ybin, T. Ling, "Research on Cloud Design Resources Scheduling Based on Genetic Algorithm" *International Conference on Systems and Informatics (ICSAI 2012)*, May 2012, Yantai, pp. 2651-2656, http://dx.doi.org/10.1109/ICSAI.2012.6223598.

[11] F. M. Aymerich, G. Fenu, S. Surcis, "A real time financial system based on grid and cloud computing" *ACM symposium on Applied Computing*, March 2009, New York, pp 1219–1220, http://dx.doi.org/10.1145/1529282.1529555.

[12] S. Liu, G. Quan, S. Ren, "On-Line Scheduling of Real-Time Services for Cloud Computing" *World Congress on Services*, July 2010, Miami, pp 459–464, http://dx.doi.org/10.1109/SERVICES.2010.109.

[13] W. Tsai, Q. Shao, X. Sun, J. Elston, "Real-Time Service-Oriented Cloud Computing" *World Congress on Services*, July 2010, Miami, pp 473–478, http://dx.doi.org/10.1109/SERVICES.2010.127.

[14] K. H. Kim, A. Beloglazov, R. Buyya, "Power-aware provisioning of cloud resources for realtime services" *International Workshop on Middleware for Grids, Clouds and e-Science*, November 2009, New York, pp.1–6, http://dx.doi.org/10.1145/1657120.1657121.

[15] K. Kumar, J. Feng, Y. Nimmagadda, Y. Lu, "Resource Allocation for Real-Time Tasks using Cloud Computing" *International Conference on Computer Communications and Networks (ICCCN)*, July 2011 Maui, pp. 1-7, http://dx.doi.org/10.1109/ICCCN.2011.6006077.

[16] G. C. Buttazzo, "Hard real-time computing systems: predictable scheduling algorithms and applications". Vol. 24. Springer, 2011, pp. 1 – 22, http://dx.doi.org/10.1007/978-1-4614-0676-1.

[17] G. Tel, "Introduction to Distributed Algorithms" Cambridge University Press, 2nd edition, 2001.

[18] E. J. H. Chang, "Echo Algorithms: Depth Parallel Operations on General Graphs" *IEEE Transactions on Software Engineering*, July 1982, pp. 391 – 401, http://dx.doi.org/10.1109/TSE.1982.235573.

[19] S. Bąk, R. Czarnecki, S. Deniziak "Synthesis of real-time cloud applications for Internet of things" *Turkish Journal of Electrical*

Engineering and Computer Sciences, to be published, http://dx.doi.org/10.3906/elk-1302-178.

[20] S. Bąk, R. Czarnecki, S. Deniziak "Synthesis of Real-Time Applications for Internet of Things" *Lecture Notes in Computer Science vol. 7719*, 2013 pp. 37-51, http://dx.doi.org/10.1007/978-3-642-37015-1_4.