Punctual Cloud

Unbinding Real-time Applications from Cloud-induced Delays

Peng, Haorui; Tarneberg, William; Fitzgerald, Emma; Kihl, Maria

Published in:
2021 International Symposium on Networks, Computers and Communications, ISNCC 2021

DOI:
10.1109/ISNCC52172.2021.9615741

2021

Document Version:
Early version, also known as pre-print

Link to publication

Citation for published version (APA):
Peng, H., Tarneberg, W., Fitzgerald, E., & Kihl, M. (2021). Punctual Cloud: Unbinding Real-time Applications from Cloud-induced Delays. In *2021 International Symposium on Networks, Computers and Communications, ISNCC 2021* IEEE - Institute of Electrical and Electronics Engineers Inc.. https://doi.org/10.1109/ISNCC52172.2021.9615741

Total number of authors:
4

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Punctual Cloud: Unbinding Real-time Applications from Cloud-induced Delays

Haorui Peng*, William Tärneberg*, Emma Fitzgerald*†, Maria Kihl*

*Department of Electrical and Information Technology, Lund University, Lund, Sweden

†Institute of Telecommunications, Warsaw University of Technology, Warsaw, Poland

{haorui.peng, william.tarneberg, emma.fitzgerald, maria.kihl}@eit.lth.se

***Abstract*—Cloud computing has become a prominent technology for the computing paradigm in various industrial sectors nowadays. For most industrial applications to perform in real-time, the support of periodic computing is required. However, it remains a challenge when the computing is executed in a cloud, since both the network connection and the cloud environment are uncertain. In this paper, we propose a new architecture to deploy real-time applications in the cloud. We call it *punctual cloud*. We detail the implementation and demonstrate how punctual cloud is deployed in a cloud-native manner on Kubernetes. We evaluate the system's performance with a real-time resource allocation problem and show that, compared to a system without punctual cloud, which has maximum 40% punctual deliveries, our proposed architecture can attain over 90% responses to be delivered punctually for the application, while also being capable of remedying the performance degradation caused by long and uncertain response delays in the system.**

***Index Terms*—Cloud computing, micorservice, real-time cloud application, cloud-induced Delays**

I. Introduction

The digital transformation driven by Industry 4.0 anticipates the integration of cloud computing into manufacturing systems. Cloud computing has ubiquitous and “infinte” computing resources and storage capacity, giving it advantages over bare-metal web servers. The integration of cloud computing allows industrial real-time applications, such as controllers and schedulers, to offload their computing tasks to the clouds. The manufacturing industry may benefit tremendously from the economic model and flexibility of cloud computing [1].

Real-time applications require timely responses from their computing tasks. Long delays in the responses can contribute to stale actuation signals to the consumer, and eventually cause system failure. These types of applications are conventionally placed near the consumer of a process, such as on an on-board and on-premises computer. However, when hosted in the cloud, longer and inevitable delays will be incurred. The network separation from the consumer, unpredictable virtualization environment, and the layered software management system can all cause extra disturbances [2]–[6]. Additionally, today's cloud services are evolving towards a microservice architecture [7], which introduces inter-communications among light weight services in an application. All these disturbances from clouds yield time-variant delays that could be greater than the desired response time of a consumer [8], [9]. Therefore, it is nontrivial to develop a viable approach for deploying real-time applications in clouds without being disrupted by the delay-inhibited environment.

Edge and fog computing were developed as dedicated cloud computing paradigms for time-sensitive processes. Fog/edge computing can reduce the response time by placing services in smaller scale cloud infrastructures that are in vicinity to the consumers. A number of studies have suggested optimization approaches to dynamically distribute computing tasks in the fog [10]–[13]. The authors of [14] proposed a dynamic switching solution between local computers and edge nodes to meet the latency requirements for a cyber-physical system. In [15], an industrial cyber-physical system for machine learning applications was deployed as a fog architecture to execute time-sensitive machine learning models.

Several studies on using clouds for real-time systems have been performed in recent years. The critical design issues for real-time services using cloud computing were summarized in [16]. The authors of [17] examined a set of cloud platforms and presented a strategy to enable cloud computing in critical control systems. The concept of control as a service was presented in [18] for autonomous vehicles, in which a controller was designed to tackle network imperfections in the system and was evaluated by a simulation study.

Even though it has been shown that the cloud can host some real-time applications, especially when having them deployed in an edge or fog architecture, there is no guarantee that the response time of a time-sensitive service is close to the one obtained by on-board computers. The time-varying delays in cloud computing systems remain a challenge. Additionally, performance evaluation on previously proposed solutions was a laborious task for academic researchers, due to the lack of well-established public edge infrastructure. Most previous studies on latency minimization in cloud computing were restricted to simulation platforms [9]–[13], [18], or to improvised edge/fog nodes, such as a single bare-metal server or on-premises controller [14], [15]. However, the disturbances from the actual execution environment, whether in the edge or a centralized cloud, have been mostly neglected.

Therefore, rather than upgrading the infrastructure to meet the stringent requirements on response time, in a previous work we have argued that real-time applications should adapt themselves to the nature of clouds [19]. To mitigate the impacts from the cloud environment, we proposed a new deployment architecture named *punctual cloud* to enable punctual response deliveries for a radio resource allocation problem over Cloud RAN [19]. This solution was evaluated and proved under a simulation environment, which however could not fully capture the characteristics of an actual cloud environment and the network.

In this paper, we adapt our punctual cloud architecture based on microservices for demanding real-time cloud applications that run at high frequency and present the implementation in a real cloud execution environment. We show in this paper that punctual cloud is infrastructure agnostic, thus it can be deployed in both edge and centralized clouds. A performance evaluation shows that, compared to a system under general deployment, which has maximum 40% punctual deliveries, our proposed architecture can attain over 90% punctually delivered responses, and it has less impact from the disturbances of the clouds and network protocols compared to the general deployment without punctual cloud.

The remainder of the paper is organized as follows. In Section II we first outline the addressed system and its properties. Based on the problem we are concerned with the system, we present in Section III briefly our proposed solution, which was investigated in [19]. In Section IV, we use an example to show in detail how this proposed architecture is adapted to the implementation of resource

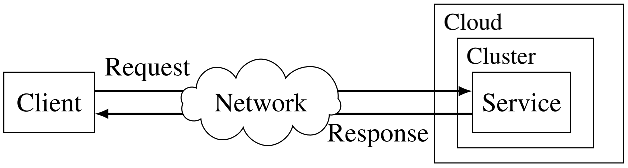

Fig. 1: System components of a general deployment.

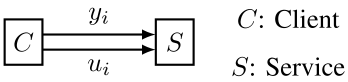

Fig. 2: System model of a cloud application under consideration.

allocation application in the cloud. In Section V and VI, we evaluate the system performance and compare our proposed architecture with a general deployment without punctual cloud.

II. System Description

In this section, we first describe the system under consideration, then present the system model and show the problems when the system is deployed in a general way.

A. Targeted system

We target a real-time cloud application consisting of a client and a service that communicate with a Request-Response pattern over a shared network. The essential components of the system are depicted in Fig. 1.

Most real-time applications such as industrial control systems, resource allocation functions and task schedulers, require computing tasks to be performed periodically at high frequency [20]. The computing tasks of such systems output actuation commands to the system actuator to keep the process running properly. The commands are usually based on the real-time state of the client. In this paper, real-time cloud application running at a deterministic frequency is under consideration. For a cloud application, the computing tasks are executed by the service which resides in the cloud, however the output signals of the tasks are delivered to the actuator co-located with the client. The requests are sent by the client to ask for actuation commands, and the responses are sent back by the service to serve the client.

The response delay of each request is highly dependent on the network connection and protocols that are utilized in the system. The choice of network connections between the client and an edge cloud depends on their relative locations, and various networks such as WiFi, mobile networks, Ethernet and even dedicated optical networks can be utilized. For the connection to a centralized cloud, the workloads of an application normally need to traverse the Wide Area Network (WAN), which induces more uncertainties on the data path between a client and its service. The unpredictable network behaviors would introduce complicated stochastic properties on the response delays of a cloud application.

As of now, the majority of cloud computing system are built on REST HTTP, since it is a comparably mature and stable solution [21]. In our targeted system, a standard deployment based on HTTP/TCP will also be utilized. We will show in Section VI that, although the response time of an application is prolonged due to the operations of TCP sessions compared to UDP, it does not contribute to a major impact on the system performance with our deployment model.

B. System model

The system can be described as a simplified model as illustrated in Fig. 2, where C represents the client and S is the service. This is a system model of a web application with general deployment. The service in the cloud is deployed in monolithic way, which is also used by the majority of cloud services.

Thus we define that the client sends outs the requests at frequency f and the interval of successive requests is $h = \frac{1}{f}$. A request sent at time i contains the state of client y_i, and it expects an actuation command u_i in response from the service. i represents the index of each time slot in the model, thus $i = \{0, 1, 2, ...\}$, and the length of a slot is h. The aim of the computing task is thus to generate the command to serve the client state at time i as:

$$u_i = \mathrm{F}(y_i) \tag{1}$$

The robustness of a real-time application relies on the promptness of each response, that is whether a response can arrive at the client before the expiration of the state updated by its request. As the client state is updated at frequency f, the maximum tolerable waiting time of each state is $h = \frac{1}{f}$.

The duration from the time when y_i is sent out until when the response u_i is delivered back to the client is defined as the *response delay* $\mathcal{D}_i$. A client state is expired if its own response cannot be delivered within h. Thus, a response that contains command u_i is called punctually delivered if its response delay $\mathcal{D}_i$ is shorter than h.

C. Problem definition

Fig. 3 illustrates an example of the potential consequences of stochastic response delays in a real-time application, which is caused by the uncertain network conditions and the cloud execution environment. In an ideal case, where the client and the service are co-located, the response delays should be near to 0, and the response is sent back immediately after a request departs. However, when the client and the service are separated via the network, together with additional execution time in the cloud, such as admission time of a proxy and workload scheduling of an ingress controller, the response delay of an application increases and becomes uncertain. When the application is running at a high frequency, even if the average response delay is shorter than the request interval h, significant numbers of the responses may not be delivered punctually due to the stochastic network properties. Relatively longer and unpredictable response delays could later result in performance degradation in the application.

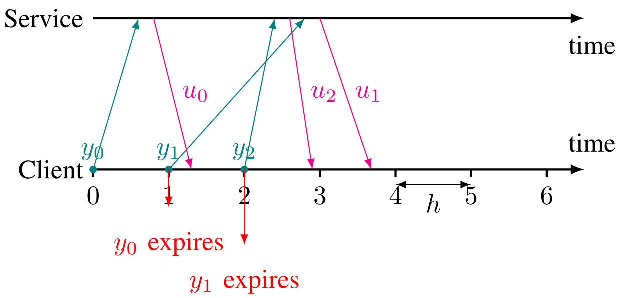

Fig. 3: Consequences of stochastic response delays on subsequent requests y_i and responses u_i.

Another important property of cloud computing is scalability, which means there can be multiple replications or threads for a service to serve many incoming requests simultaneously. The advantage of having replications is that, when having two successive requests y_i and y_{i+1}, the computing task of the latter one can be performed before the previous task is finished, and the queuing time of the requests at the server side is significantly mitigated. However, a problem raised by this property is that the responses of the requests may arrive out of order. Under such circumstances, the actuation command given by a response might not be executed at the proper time it is intended to be.

Therefore, it is important for real-time applications to be unbound from the delays incurred by the clouds, which means no matter how long or how variable the

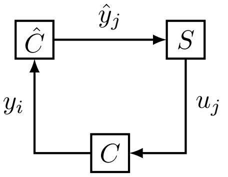

Fig. 4: System model with punctual cloud architecture.

response delays are, the system needs to guarantee that most responses are delivered in a timely fashion and the underlying actuation commands can serve the states at the time they are intended to.

III. Proposed Punctual Cloud Architecture

In this section, we present a new system architecture that addresses the requirement for real-time cloud applications, which is named as *punctual cloud*. This architecture requires no upgrading on the raw cloud infrastructure, which means the network connection and properties remain the same as the general deployment of a cloud application. But the application can adapt itself to the network and environment dynamics and enable punctual response deliveries. We also show in the next section how the application is implemented in the cloud using the example of the resource allocation problem investigated in [19].

The purpose of punctual cloud is to remedy the impact of stochastic response delays caused by the network and unpredictable cloud environment. Whether a response is punctually delivered is decided by its actuation command u_i. If u_i can serve client state at time i, the response can be considered as punctually delivered. Therefore, in a case when the response delay $\mathcal{D}_i$ is inevitably longer than the longest tolerable waiting time h of a request that contains state y_i, one solution is to return with an actuation command u_j in the response, which pertains to a future state y_j and $j - i \geq h$.

The system model of the application with punctual cloud architecture is illustrated in Fig. 4, in which $u_j = \mathrm{F}(\hat{y}_j)$ and $\hat{y}_j$ is an estimated state for a future time slot j. In this new system model, both the estimator $\hat{C}$ and the service is deployed in the cloud. To generate u_j, $\hat{C}$ needs to estimate:

1) The time j when the actuation command serves the client state.
2) The client state $\hat{y}_j$ at time j.

The proposed method is agnostic to applications, but it requires the future client state to be estimated. In this manner, a real-time application that can be deployed needs to have sufficiently accurate estimation on its client states, which are the inputs to the computing tasks of the service in the cloud. Example applications can be channel estimation based on user mobility prediction in mobile networks, or a control system that has the controller deployed remotely in the cloud, provided that the system models are well developed to make estimations.

IV. Implementation

In this section, we will give an example based on the resource allocation problem investigated in [19], which addresses a radio resource allocation problem for massive MIMO. The client is the Remote Radio Head (RRH) of a Cloud RAN system that assigns uplink pilots to the end-users of a wireless system [22]. This real-time application requires punctual responses at a high frequency from the cloud service. We will provide the details of how the application with the proposed model is implemented in a cloud-native way on Kubernetes[1], which is an open-source platform for deploying and orchestrating containerized applications in the cloud.

[1] https://kubernetes.io

A. Example resource allocation problem

In the example resource allocation problem, the client can be seen as a resource scheduler that assigns resources to a set of users who ask for them. Assuming a user demands resource at frequency g, each demand requires the user to consume one resource. If not enough resources are assigned to be consumed by the user and a demand has been pending for $\frac{1}{g}$, we call it the *failure* of this demand. If a resource is assigned to a user but there is no pending demand from this user at this moment, we call this a resource *waste*.

In the example problem, we assume there are N users, each is running at frequency g_n to demand one resource each time, where $n \in \{1, 2, ...N\}$. The client C is a resource scheduler and releases P resources each time at frequency f, where $g_n < f$ for all users. Therefore, the client state y_i that is sent as a request to the service is the number of pending demands at time i from each user. Denoting that the number of pending requests from the nth user at time i is q_{n_i}, the client state at time i can be denoted as:

$$y_i = [q_{1_i}, q_{2_i} ... q_{N_i}]^T \tag{2}$$

The service in the cloud of the application makes a response containing a command u_i to the resource scheduler, deciding which user can be assigned a resource at time i, thus we define:

$$u_i = [p_{1_i}, p_{2_i} ... p_{N_i}]^T \tag{3}$$

where:

$$p_{n_i} = \begin{cases} 1 \text{ assign a resource to user } n \\ 0 \text{ no resource is assigned to user } n \end{cases} \tag{4}$$
$$\sum_{n=1}^{N} p_{n_i} \leq P$$

Under the general system model shown in Fig. 2, the service computes u_i based on each delivered state y_i. However, due to the response delay, the actuation command may not be applicable to the new state at the client when it is delivered. Therefore, it requires that the actuation command is capable of serving a future state y_j, so that it can be delivered before its intended state comes.

B. Example problem under punctual cloud architecture

In our proposed architecture, the actuation command is computed as $u_j = \mathrm{F}(\hat{y}_j)$, and $\hat{y}_j = [\hat{q}_{1_j}, \hat{q}_{2_j} ... \hat{q}_{N_j}]^T$ is an estimated state of the client at time j. The goals of u_j computed by each task in the proposed system model become:

1) Guarantee that u_j is delivered on time, implying $D_j \leq j - i$.
2) The failures of user demands and resource waste should be mitigated compared to the general system deployment.

Function F in Eq. (1) is a scheduling strategy that maps the number of pending demands to an allocation decision. Since optimizing the scheduler is not the main goal in this paper, we simply choose a fair allocation strategy, in which P users who have the most pending demands can get assigned a resource for each.

As we mentioned in Section III, the computing task is triggered by the client state y_i in the request but computes the allocation command u_j. Both time j when u_j can be executed and the client state $\hat{y}_j$ need to be estimated. In order that the estimations can be made, the following information about the client at time i is also sent to the service:

1) The response delay $\mathcal{D}_{i-1}$ of the last delivered response.
2) The ratio of punctually delivered responses before i, denoted by r_i.
3) The interval of the most recent successive demands from each user $\frac{1}{g_{n_i}}$ for $n \in \{1, 2, ...N\}$.

4) The time when the most recent pending demand was released by each user, noted as $\{t_{ni}\}$ for $n \in \{1, 2, ...N\}$.

To predict the time j when the allocation command u_j can be actuated, we first estimate the average response delay $\hat{\mathcal{D}}$ of the application, which is made based on the delay of the latest delivered response $\mathcal{D}_{i-1}$ with Exponential Moving Average (EMA) and its weight ω:

$$\hat{\mathcal{D}} = \omega\hat{\mathcal{D}} + (1-\omega)\mathcal{D}_{i-1} \tag{5}$$

The actuation time j is then calculated based on the average delay plus an offset σ, which is used to compensate the jitter of the response delay distribution:

$$j = i + \lceil\frac{\hat{D}}{h}\rceil + \sigma \tag{6}$$

The offset σ is initialised to 0 and is decided by a step controller that retains the ratio of punctually delivered responses r_i above a bounding value R:

$$\sigma = \begin{cases} \sigma + 1 & \text{if } r_i < R \\ \sigma & \text{otherwise} \end{cases} \tag{7}$$

The frequencies of making resource demands by each user can also be estimated with EMA and weight ω:

$$\hat{g}_n = \omega\hat{g}_n + (1-\omega)g_{ni} \tag{8}$$

Once frequency $\hat{g}_n$ is estimated, along with the predicted actuation time j and the most recent demand time t_{ni}, one can estimate whether user n will release a new resource demand at time j by checking if an integer k exists and gives:

$$(j-1)h < t_{ni} + \frac{k}{\hat{g}_n} \leq jh \tag{9}$$

If such an integer k exits, it is most likely that user n will release a new resource demand at time j, thus the client state $\hat{y}_j$ can be estimated as 1. Otherwise, it is estimated as 0. An overview of estimating $\hat{y}_j$ and generating u_j based on the inputs is given in Algorithm 1.

Algorithm 1 Estimation on $\hat{y}_j$ and generating u_j in the cloud

```
Init: $\omega, R, \hat{\mathcal{D}}, \{\hat{g_n}\}, \{\hat{t_n}\}, \sigma = 0$
Input: $y_i, \mathcal{D}_i, r_i, \{t_{ni}\}, \frac{1}{g_{ni}}$
 1: $\hat{D} \Leftarrow \omega\hat{D} + (1-\omega)D_i$
 2: if $r_i \leq R$ then
 3:     $\sigma \Leftarrow \sigma + 1$
 4: end if
 5: $j \Leftarrow i + \lceil\frac{\hat{D}}{h}\rceil + \sigma$
 6: for all $n \in \{1, \ldots, N\}$ do
 7:     $\hat{g}_n \Leftarrow \omega\hat{g}_n + (1-\omega)g_{ni}$
 8:     for all $k \in \{0, 1, 2, ...\}$ do
 9:         if $(j-1)h < t_{ni} + k/\hat{g_n} \leq jh$ then
10:             $\hat{q_{nj}} = 1$
11:             break
12:         end if
13:     end for
14: end for
15: $\hat{y}_j \Leftarrow \{\hat{q_{1j}}, \hat{q_{2j}}...\hat{q_{nj}}\}$
16: $u_j \Leftarrow \mathrm{F}(\hat{y}_j)$                    ▷ Fair allocation
Output: $u_j$
```

C. Implementation on Kubernetes

The implementation of the punctual cloud system architecture is illustrated as Fig. 5. The application can be implemented in a microservice architecture with two different services being deployed. The "Allocation Service" runs the same task as the general deployment shown in Fig. 1, which is responsible for making allocation decisions using the function on Line 16 of Algorithm 1. The "Estimation Service" is an additional service compared to the general deployment, and it runs the computation of Lines 1-21 in Algorithm 1. Both services are containerized and deployed in a Kubernetes cluster, and each service can run in several Pods, which are the endpoints of the workloads that perform the

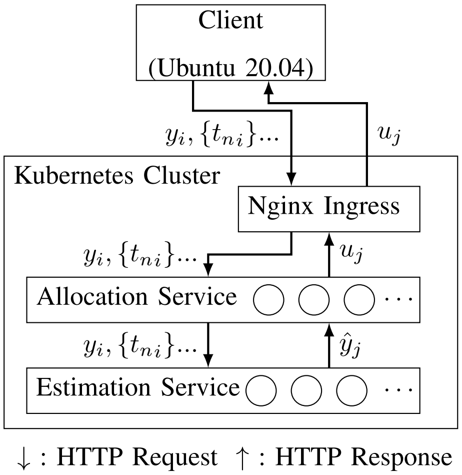

Fig. 5: Implementation of the new system model on Kubernetes

computation of each service. Traffic from the client is load balanced by a Nginx ingress controller[2] of the cluster and directed to the Allocation Service, which however, passes on the state information to the Estimation Service via a new HTTP request. The Estimation Service estimates the average response delay and the client state; it then returns a response with the estimated state $\hat{y}_j$, so that the Allocation Service can generate the allocation command u_j and respond to the client.

A simulation of the users demanding resources is running on the client along with the allocation actuation process. The client also establishes a HTTP session at a frequency of f to the service in the cloud and sends out a request with current state information about the users' pending demands.

In the example implementation, a Nginx ingress controller is utilized as a load balancer of the application. But the Allocation Service can also be deployed in other ways such as NodePort, in which no ingress controller is required. Also, the choice of ingress relies on the actual cloud environment; other type of load balancers can be used if supported by the cloud provider.

Moreover, the Estimation and Allocation services in Fig. 5 communicate via HTTP sessions, but it is easy to add a service mesh layer such as Istio[3] and allow Envoy proxies to intercept and direct the traffic between services. Other protocols such as gRPC can also be deployed instead of HTTP. Both implementation examples of the same application with and without the service mesh are provided on GitHub [23].

In this implementation, the Allocation Service and Estimation Service are deployed separately. The Allocation Service is unchanged from the general system deployment except for its communication with the Estimation Service. In this manner, one can easily adapt the architecture to any type of real-time applications without changing the original deployment, but only needs to add suitable estimation or optimization solutions in the Estimation Service.

V. Evaluation

In this section, we describe our experiment setup based on the implementation presented in Section IV. The objective of the experiments was to evaluate our proposed punctual cloud architecture and its implementation. We compare the punctual cloud to the general system deployment shown in Fig. 2, which has no estimation and runs only an allocation service that is deployed in monolithic architecture in the cloud. The allocation service computes u_i based on incoming state information y_i in each request from the client.

To evaluate the system performances under a more controllable environment than a public cloud, we deployed the services on a bare-metal Kubernetes cluster that has 7 nodes of Ubuntu 20.04. The client machine

[2] https://www.nginx.com/products/nginx-ingress-controller/

[3] https://istio.io

runs the simulation of the resource allocation problem, and simulates the traffic generation of N users, where each demands resources at frequency g_n. Meanwhile the demands are served by allocation commands from the service in the cluster also at frequency f. The client and the cluster are connected via 1Gbps Ethernet. The round-trip time between the client and the master node of the cluster measured by ICMP is 0.215ms, which is negligible compared to the frequency that the application runs at.

In order to have network properties close to WANs, we used Netem[4] on the client machine to emulate various network properties by regulating delay mean and jitter values under a Pareto distribution. While having the application running at frequency f, and the interval of two successive client requests as $h = \frac{1}{f}$, we set the Netem delay with mean value μ varying from 0ms to $0.8h$, and jitter value σ varying from 0 to μ under each delay configuration.

We note that, although the added delay and jitter in the Netem configuration is smaller that the interval h, the actual response delays also depend on the TCP operation, admission time to the cluster and internal traffic routing within the cluster. These operations ultimately give response delays two to three times longer than the added delay from the Netem. Moreover, the average response delay returned the punctual cloud architecture is about 11ms longer than the general deployment. This is caused by additional communications between the two services within the cluster, while the general deployment only has one service.

In Table I, we show the experiment parameters used in the simulation. All of the parameters could be adjusted according to the deployed applications. For the evaluation, we ran the experiments under each parameter configuration for 10 repetitions, and each experiment was run for 12.5 minutes.

TABLE I: Experiment parameters used in the evaluation

Symbol	Parameter Definition	Value
f	Frequency of sending request by the client	-
h	Interval of client requests $h = \frac{1}{f}$	50ms
μ	Netem delay configuration value	-
σ	Netem jitter configuration value	-
N	Number of user having resource demands	10
g_n	Frequency of consuming resources by user n	$\frac{1}{70\text{ms}}$
	Max tolerable waiting time user n demands	
P	Number of resources released by the client every h	12

We evaluated the performance of the resource allocation application from two perspectives:

1) *Punctual deliveries*: This is the property required by all kinds of real-time applications. It is defined as the ratio of punctually delivered responses to the total number of requests sent by the client.
2) *Failures and Waste*: They are two properties representing the performance of the allocation problem, which is impacted by *Punctual deliveries*. The term *Failures* is short for user demand failures, which is the ratio of failed demands to all the demands from all users. *Waste* is short for resource waste, which is the ratio of wasted resources to the total number of resources allocated by the applied actuation commands.

VI. Results

In this section, we present and discuss the results of our experiments. We show with Fig. 6 and 7 that our proposed punctual cloud architecture significantly improves the application performance regarding the three properties we addressed in the resource allocation problem, as it guarantees that the responses of a cloud application are punctually delivered even under long and

[4] https://wiki.linuxfoundation.org/networking/netem

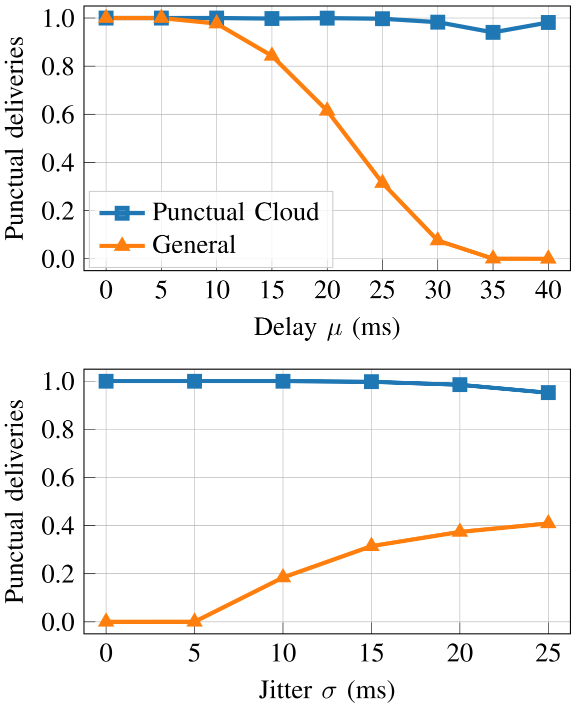

Fig. 6: Punctual deliveries among all the responses for experiments under given Netem configurations.

uncertain delays. All figures in this section illustrate the performance under the same Netem configuration. The results from the delay evaluation were made with Netem delay value μ varying from 0 to 40ms and jitter σ was set to 0.6μ, implying than the coefficient of variance of the delay in the Netem configuration is 0.6. The show results of the evaluation with varying jitter were made when the Netem delay value was fixed to $\mu = 25$ms, and the jitter σ varied from 0 to μ. The actual mean response delay also increased as the jitter configuration increased under the same Netem delay value. The values of other experiment parameters of the shown figures are presented in the third column of Table I. The standard deviation of all results is small enough so that they are not displayed in the figures.

A. *Punctual deliveries*

In the general system deployment, when the response delay is longer than the request interval h, the response is not delivered punctually, and the client state expires before being served by the actuation command. In contrast, in the system deployed in punctual cloud architecture, the actuation command in each response may be intended to serve a future state, which will be carried by a request that has not yet been made. Thus when a response is delivered before the expiration of the state it is intended to serve, it is punctually delivered.

Fig. 6 shows that when jitter in the delay increased, the general deployment got more punctually delivered responses. This is related to the empirical cumulative function of the response delay measurements and the cumulative probability of value h. For example, in the case when the jitter value $\sigma = 0.4\mu$ and $\mu = 0.5h = 25$ms in the Netem configuration, the cumulative probability of h is 24.7% from the experiment. However, when $\sigma = 0.8\mu$, the cumulative probability becomes 40.5%. This explains why the number of punctual deliveries is greater when the jitter of the response delays is larger. In all cases, the performance of punctual deliveries is more robust in our proposed architecture as the delay mean μ or the jitter σ increases.

B. *Failures and Waste*

As for the resource allocation problem, a stale actuation command will be executed if the expected response is not punctually delivered. In such a case, the stale command is however computed based on an expired client state and may not meet what the current state demands. It may further cause failures and resource waste due to false allocation. Thus, when an application has fewer punctual delivered responses, it is more likely that the other performance properties will be impacted.

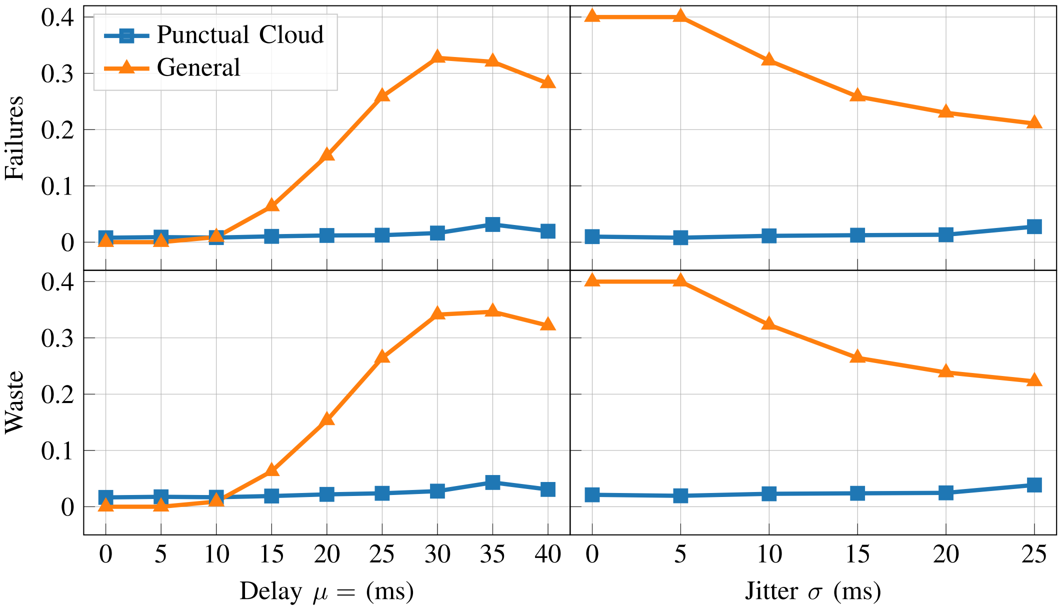

Fig. 7: Demand failures and resource waste of the resource allocation under given Netem configurations.

Fig. 7 depicts the performance for allocation failures and resource waste with the two different methods. As we can see from the figures, when the response delays are shorter than the request interval h, there is no significant difference in the allocation failures and resource waste. The general deployment even had better performances when the response delay was small enough that most responses can be punctually delivered. This implies that the allocation performance also depends on the accuracy of the client states estimation. However, when delay was much longer, and the ratio of punctually delivered responses decreased, the estimation in the punctual cloud can significantly compensate the impact brought by long delays.

When the jitter in the delay increased, we can see from the sub-figures in the left column of Fig. 7 that the performance of resource allocation in the general deployment was better, since the punctual deliveries were higher (shown in Fig. 6). The performance in the punctual cloud architecture had less impact from larger jitters, since the punctual deliveries do not completely rely on the distribution of the response delays in this case.

Furthermore, even though stale commands were mostly adopted in the general deployment in some cases, the allocation performance was not degraded completely. This is because a stale response does not mean absolute false allocation. If frequency g_n is larger, there will be more frequent demands on resources from the users, and it will also lead to less resource waste.

VII. Conclusions

Real-time cloud applications rely heavily on low latency in the response time. However, the requirements of some demanding applications are not pledged by the stochastic properties of the network and unpredictable cloud environment. In this paper, we proposed the punctual cloud architecture to implement real-time applications in the cloud. With our proposed model, the cloud service computes and makes responses based on the estimation of response delays and client states. In the evaluation, we showed that the application in our proposed model can provide punctual responses to the client. With more responses punctually delivered, it can also improve the performance of the application, such as a resource allocation process.

Our proposed architecture promises to compensate the performance degradation caused by long response delays of a real-time application. If the network and cloud infrastructure can guarantee low latency without any disturbances, and the response delay always meets the client requirements, there is no pressing need to introduce additional services and a longer data path to remedy the other performance properties of the application with the proposed architecture. In such a case, the performance should be close to the ideal case, wherein the client and computing tasks are co-located.

In the case of edge computing, the response time of cloud applications can mostly meet the stringent latency requirements even in a general deployment. However, the uncertainties in the network would cause the responses delay to experience an abrupt rise at certain times. Our future work is to develop an adaptive model that switches between the deployment with and without punctual cloud, based on the network condition in real-time.

Acknowledgement

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the SEC4FACTORY project, funded by the Swedish Foundation for Strategic Research (SSF), and the 5G PERFECTA Celtic Next project funded by Sweden's Innovation Agency (VINNOVA). The work of Emma Fitzgerald was also supported by the National Science Centre, Poland, under the grant no. 2019/35/D/ST7/00350 "Quality of Service in IoT Applications Using Large Scale Antenna Systems". The authors are part of the Excellence Center at Linköping-Lund on Information Technology (ELLIIT), and the Nordic University Hub on Industrial IoT (HI2OT) funded by NordForsk.

References

[1] X. Xu, "From cloud computing to cloud manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 1, pp. 75–86, feb 2012.

[2] M. García-Valls, T. Cucinotta, and C. Lu, "Challenges in real-time virtualization and predictable cloud computing," *Journal of Systems Architecture*, vol. 60, no. 9, pp. 726–740, oct 2014.

[3] S.-H. Ha, D. Venzano, P. Brown, and P. Michiardi, "On the impact of virtualization on the I/O performance of analytic workloads," in *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*. IEEE, may 2016, pp. 31–38.

[4] G. Wang and T. S. E. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *2010 Proceedings IEEE INFOCOM*. IEEE, mar 2010.

[5] U. Drepper, "The cost of virtualization: Software developers need to be aware of the compromises they face when using virtualization technology." *Queue*, vol. 6, no. 1, p. 28–35, Jan. 2008.

[6] L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and M. Kihl, "Impact of etcd deployment on kubernetes, istio, and application performance," *Software: Practice and Experience*, vol. 50, no. 10, pp. 1986–2007, 2020.

[7] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, *Microservice Architecture*. O'Reilly Media, Inc., 2016.

[8] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems - MMSys '10*, New York, USA, 2010.

[9] S. Maheshwari, D. Raychaudhuri, I. Seskar, and F. Bronzino, "Scalability and Performance Evaluation of Edge Cloud Systems for Latency Constrained Applications," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, oct 2018, pp. 286–299.

[10] M. S. Elbamby, M. Bennis, and W. Saad, "Proactive edge computing in latency-constrained fog networks," in *2017 European Conference on Networks and Communications (EuCNC)*. IEEE, jun 2017.

[11] J. Zhang, X. Hu, Z. Ning, E. C. Ngai, L. Zhou, J. Wei, J. Cheng, B. Hu, and V. C. M. Leung, "Joint Resource Allocation for Latency-Sensitive Services Over Mobile Edge Computing

Networks With Caching," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4283–4294, jun 2019.

[12] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Transactions on Internet Technology*, vol. 19, no. 1, 2018.

[13] C. Shi, Z. Ren, K. Yang, C. Chen, H. Zhang, Y. Xiao, and X. Hou, "Ultra-low latency cloud-fog computing for industrial Internet of Things," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, apr 2018.

[14] Y. Ma, C. Lu, B. Sinopoli, and S. Zeng, "Exploring Edge Computing for Multitier Industrial Control," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3506–3518, nov 2020.

[15] P. O'Donovan, C. Gallagher, K. Bruton, and D. T. O'Sullivan, "A fog computing industrial cyber-physical system for embedded low-latency machine learning Industry 4.0 applications," *Manufacturing Letters*, vol. 15, pp. 139–142, jan 2018.

[16] W.-T. Tsai, Q. Shao, X. Sun, and J. Elston, "Real-Time Service-Oriented Cloud Computing," in *2010 6th World Congress on Services*. IEEE, jul 2010, pp. 473–478.

[17] P. Skarin, W. Tärneberg, K.-E. Arzen, and M. Kihl, "Control-over-the-cloud: A performance study for cloud-native, critical control systems," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE, dec 2020, pp. 57–66.

[18] H. Esen, M. Adachi, D. Bernardini, A. Bemporad, D. Rost, and J. Knodel, "Control as a service (CaaS)," in *Proceedings of the Second International Workshop on the Swarm at the Edge of the Cloud*. New York, USA: ACM, apr 2015, pp. 13–18.

[19] H. Peng, W. Tärneberg, and M. Kihl, "Latency-aware radio resource allocation over Cloud RAN for industry 4.0," in *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, jul 2021.

[20] P. Rygielski, V. Simko, F. Sittner, D. Aschenbrenner, S. Kounev, and K. Schilling, "Automated Extraction of Network Traffic Models Suitable for Performance Simulation," in *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*. New York, NY, USA: ACM, mar 2016, pp. 27–35.

[21] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration," *ACM Comput. Surv.*, vol. 51, no. 6, Jan. 2019.

[22] E. Bjornson, E. G. Larsson, and M. Debbah, "Massive MIMO for Maximal Spectral Efficiency: How Many Users and Pilots Should Be Allocated?" *IEEE Transactions on Wireless Communications*, vol. 15, no. 2, pp. 1293–1308, feb 2016.

[23] H. Peng, "latency-aware-cloud-scheduler," Github. [Online]. Available: https://github.com/HaoruiPeng/cloud-scheduler-mesh