# Control over the Edge Cloud - An MPC Example

Årzén, Karl-Erik; Skarin, Per; Tärneberg, William; Kihl, Maria

2018

*Document Version:*
Publisher's PDF, also known as Version of record

Link to publication

*Citation for published version (APA):*
Årzén, K.-E., Skarin, P., Tärneberg, W., & Kihl, M. (2018). *Control over the Edge Cloud - An MPC Example*. Paper presented at 1st International Workshop on Trustworthy and Real-time Edge Computing for Cyber-Physical Systems, Nashville, Tennessee, United States.

*Total number of authors:*
4

# Control over the Edge Cloud - An MPC Example

Karl-Erik Årzen*, Per Skarin*†, William Tärneberg‡, and Maria Kihl‡
* Department of Automatic Control, Lund University, Sweden
†Ericsson Research, Lund, Sweden
‡Department of Electrical and Information Technology, Lund University, Sweden

***Abstract*—The distributed cloud with edge data centers can be used to host mission-critical control applications. In the paper an MPC control approach is used. A test bed that allows physical experiments using a 5G base station, an edge node, and two remote data centers is presented.**

## I. Introduction

The future networked society, empowered by hundreds of billions of connected devices, will fundamentally change the way we need and do compute. Already today, the limited computing and storage capacity of end-user and Internet of Thing devices - such as smart phones, laptops, cameras, and sensors - are complemented by remote data centers. However, there are limitations on what kind of interactive and real-time applications that can be deployed on today's cloud due to its inability to provide guaranteed end-to-end performance with low communication delay and little jitter in that delay. A consequence of this is, e.g., that closed-loop control applications with hard real-time constraints currently cannot be deployed in the cloud. One major inhibitor is the delay in excess of 50 ms incurred by today's wireless radio technology, i.e., 4G/LTE. However, with the introduction of 5G, the radio access delay is poised to drop to a few stable milliseconds, removing this hurdle. Still, latency between the base station and the remote data center will typically be too long for many applications. Fog computing is a proposed remedy to this problem. A fog computing infrastructure is hyper-distributed and resource heterogeneous, ranging from user-near data centers at the edge to traditional distant data centers. The edge data centers are typically small data centers associated with a base station, a radio cell, a production plant, some transport system infrastructure, or an office building, see Fig. 1. In the fog, software applications can be dynamically deployed in all types of network nodes to meet their individual performance goals. A prominent use case for edge data centers are latency sensitive closed-loop control applications, e.g., in process control and automation, industrial robotics, or traffic control. Another use case that is actually the driver for the distributed cloud is the ongoing network function virtualization that takes place in the telecommunication network itself. Instead of using dedicated machines to implement different network functions, e.g., firewalls, load-balancers, or intrusion detection devices, one aims to implement them as software modules executing in virtual machines or containers on top of cloud computing infrastructure. The closer the virtualized network functions comes to the radio base band processing the larger the need for local data centers becomes. Once these data centers are in place it is not unlikely that excess capacity will be sold using similar business models as what is used in today's cloud. The contribution of this position paper is the presentation of a use case for edge data centers: mission-critical control over the cloud. Model Predictive Control (MPC) [1] is a control technique that fits naturally for implementation in the cloud. A 5G-based fog computing test-bed is briefly described with which one can conduct real-time experiments with MPC over the cloud including support for dynamic migration of MPC controllers between the different nodes in the distributed cloud using the Internet of Things (IoT) framework Calvin [2]. For more details about the test-bed see [3].

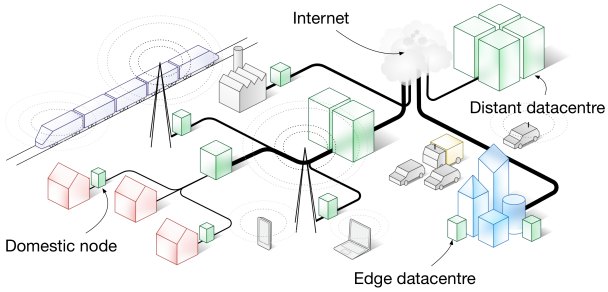


Fig. 1. Distributed fog cloud overview.

## II. Distributed Cloud Characteristics

In the distributed cloud compute and storage services are provided in every node from the remote data center to the edge data center in addition to what is provided by the end-user device. The typical control-related characteristics of this scenario are as follows. The closer to the remote data center, i.e., "the further up in the sky", the longer the communication delay will be, the shorter the computational delay will be (more powerful servers), and the larger the capacity will be (more servers). From a control performance point of view deploying the controller in the cloud typically involves a trade-off between the increased communication delay and the decreased computational delay. However, deploying the control computations in the cloud also has other characteristics than the more latency and performance related. An advantage could be the potential access to additional sensor data and more and better models, e.g., obtained using machine learning. A disadvantage could be that the potential points of failure increase and that, due, e.g., to failures or contention, the computations may have to be migrated to some other node in the distributed cloud with different latency characteristics.

## III. Control over the Cloud

A natural question to ask is what type of control that may benefit from cloud deployment. In industrial process control the basic control is often provided by Proportional, Integral, and Derivative (PID) controllers. These controllers require very few computations, e.g., around 15-20 lines of C code for a good PID controller including logical safety guard code. Since they typically also guarantee a basic level of performance, safety, and stability it is not recommendable to place this in the cloud. Instead it is the control and planning that is performed at the supervisory control level and above, that are the natural candidates for cloud deployment. In this paper the focus is Model Predictive Control (MPC), a common supervisory controller type. In an MPC the control signal is obtained through the periodic solution of an optimization problem each time new sensor data is available. The advantages of MPC are that it is a multi-input, multi-output (MIMO) controller and that it is able to guarantee that the control signal and the process state fulfill user-defined constraints, e.g., constraints on the maximum and minimum values of the control signals. The disadvantage is that it is compute-intensive and that the execution time can vary substantially, typically more than one order of magnitude, from one invocation of the MPC to the next. This typically has to do with whether the constraints are active or not. The output of the MPC, i.e., the control signals, are used as reference or setpoint signals for the underlying basic controllers. Each invocation of the MPC generates a sequence of control signals, consisting of the control signal(s) to apply at the current sampling instant, at the next sampling instant, etc all the way up to the control signal to apply at the sampling instant given by the current sampling instant plus the horizon. Only the first of these signals is sent to the process and then the same procedure is repeated at the next sampling instant. The input to the MPC is in most cases the process state. Typically an observer, e.g., a Kalman filter, is used to estimate the state vector. Mathematically the MPC controller can be expressed as

$$\underset{u_0,u_1,\ldots}{minimize} \quad \sum_{t=0}^{T-1} L(x_t,u_t) + \phi(x_T) \tag{1}$$

$$s.t. \quad x_{t+1} = f(x_t,u_t) \tag{2}$$

$$u_t \in U, x_t \in X \tag{3}$$

where $L(x_t, u_t)$ is the cost function that should be minimized. The function $\phi(x_T)$ assigns a different value specification to the final (or terminal) state $x_T$. The number of time steps $T$ is called the prediction horizon and specifies how far into the future the controller anticipates control actions. $f(x_t, u_t)$ is the process model and (3) is a set of expressions which state limitations to the plant inputs and the state space. In the case of a quadratic cost function, a linear process model, and linear constraints, the resulting optimization problem is convex. For convex optimization problems efficient solvers are available. However, the problem with large execution time variations is still present. In the case of, e.g., a nonlinear process model the optimization problem becomes non-convex. Also in this case solvers are available but they in general provide less formal guarantees on optimality etc.

### A. Use Case: Process Control

One use case is process control. Here we assume that an MPC controller is used for supervisory control and that it normally is executing in an edge data center and that wireless communication is used. If for some reason connectivity is lost the MPC calculation must be migrated from the edge data center to the local server, i.e., a backup MPC implementation must be available in the local server. This type of migration is an example of a vertical handover. However, due to lower capacity it might not be possible to solve the same MPC problem here. Several possibilities then exist. One is to solve the same optimization problem, but less often, i.e. use a longer sampling period for the MPC. Another possibility is to solve a smaller problem, e.g., only covering the most economically important parts of the plant. When the connectivity is restored the MPC can be migrated back to the edge data center. Another cause for migration could be contention in edge data center or simply a fault in the edge data center. In that case the MPC could either be migrated to the local server as before or be migrated further up, e.g., all the way to the remote data center.

### B. Use Case: Fuel Optimization for Trucks

Optimization of fuel consumption for trucks is a problem that can be approached by MPC. Here the aim is to calculate the optimal velocity for the truck that minimizes the fuel consumption taking both geographical map information and dynamic traffic information into account. One could envision that the resulting MPC problem is too complex to be solved completely on an on-board ECU. Hence, the MPC. or part of it, need to placed in the edge data center. However, as the truck moves from one cell to another the MPC needs to follow, i.e., a horizontal handover should be done from the previous edge data center to the edge data center that is now closest to the truck. Also in this use case lost connectivity and poor network coverage need to be handled. In that case the optimization calculations have to be performed in the truck most likely using a simpler formulation.

## IV. A 5G-Based Fog Computing Testbed

In order to evaluate the proposed approach a physical test bed has been developed. An overview of the test bed is shown in Fig. 2. The test bed consists of four main parts. As the physical plant to be controlled a ball and beam process is used. The control objective is to move to and keep the ball at a desired reference position on the beam. The ball position and the beam angle are measured using sensors. The mathematical model for the process is a fourth order linear model with the beam angle, beam angular velocity, ball position, and ball velocity as the states. A Raspberry Pi adjacent to the process is used as the local server. It communicates with the cloud via the Lund Massive MIMO (LuMaMi) 5G base station [4], [5]. Massive Multiple Input Multiple Output (MIMO) is the

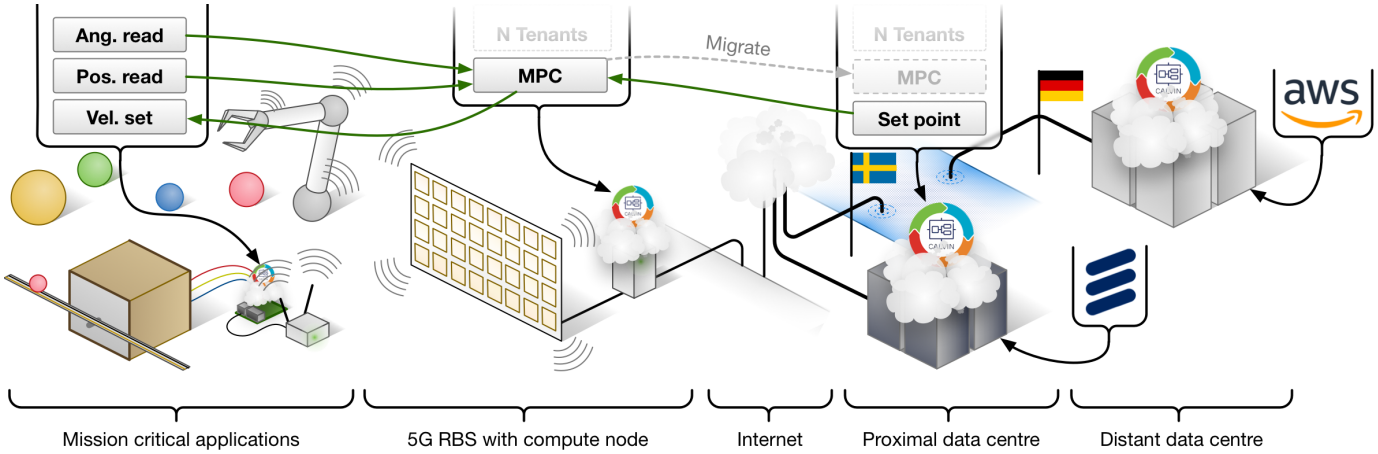


Fig. 2. System overview, from [3]

emerging Radio Access Technology (RAT) for 5G. Fundamentally, massive MIMO is a Multi-User MIMO scheme, which can simultaneously communicate with multiple devices on the same wireless resource. Additionally, massive MIMO operates with significantly more antennas than existing 4G/LTE-based RATs (150 for LuMaMi). Massive MIMO's spectral efficiency is a few orders of magnitude greater than existing RATs. The increased spectral efficiency can be used towards serving more simultaneous devices, increase the throughput, or realizing massive Machine Type Communication where a large number of devices can be reliably served simultaneously at a latency $\leq$ 5ms. A Linux server running with the PREEMPT_RT patch set [6] directly connected to LuMaMi is used to represent the edge data center. The system also consists of the Ericsson Research Data Center (ERDC) situated a few kilometers away from the cell. There we run on top of Open Stack Pike and our instance (a c4m16) has four Intel i7 cores registered by Linux as 1.6 Ghz, and 16 GB of RAM. Finally the system also consist of Amazon Web Services (AWS) data center in Frankfurt where a EC2 instance (a c4.large) with two Intel Xeon cores at 2.9 Ghz and 8 GB of RAM is used.

### A. Calvin

As the cloud software platform we use Calvin. Calvin is distributed, event-driven, server-less, and is based on a dataflow programming model. The operational units of Calvin are called actors (nodes in data-flow) while a runtime is an instantiation of the Calvin application environment on a device. In our present implementation there is a one-to-one mapping between Calvin runtimes and compute nodes and we therefore interchangeably refer to them simply as nodes. An actors' input and output messages, are known as tokens. A set of actors and their interconnections constitute an application. Actor states can be migrated and horizontally scaled across nodes.The Calvin framework can autonomously migrate and place actors to load-balance nodes and to meet its own performance goals. However, application owners can specify requirements for actors which tie them to a preferred runtime. For example, a sensor reading actor can be required to be placed on the node associated with the physical plant.

### B. Experimental Evaluation

In order to evaluate the approach an MPC controller for the ball and beam process was developed using the QPgen solver [7]. Through Calvin this MPC controller can be migrated between the different nodes in the system. To characterize and verify the basic functionality of the system we ran the MPC on each of the nodes. With each test we let the MPC control the beam for 60 minutes while alternating the set-point of the ball between the centre position and one side of the beam. To be robust in this experiment, we used a large margin to the

| Node | RTT (ms) | MPC (ms) | Aggr. delay (ms) |
|---|---|---|---|
| Plant | - - - | 5, 5.2, 10 | 15, 15, 25 |
| Edge | 9, 10, 12 | 0.8, 1.05, 1.2 | 55, 80, 115 |
| ERDC | 12, 13, 18 | 0.1, 1.1, 1.5 | 60, 82, 130 |
| AWS | 26, 28, 35 | 0.65, 0.65, 0.7 | 100, 130, 225 |

end of the beam, meaning essentially that the likelihood that the constraints would become active was small. The resulting time measurements are shown in the table above. The first column contains the network round-trip time (RTT) from the Raspberry Pi (Plant) to the other nodes. The second column shows the MPC execution time and the third the aggregated latency including both the communication latency, the MPC latency, and the overhead caused by the software platform. Each cell contains three numbers: the 5% value, the median, and the 95% value; all in milliseconds. For a full box plot see [3]. As seen in the table the wireless link introduces a 5ms oneway latency and the Raspberry Pi at the plant is many times slower than the other systems. The AWS node is faster than the edge node and ERDC which is to be expected. It can also be seen that a significant proportion of the delay in the system is introduced by the software platform and not the network. In Fig. 3 we dynamically migrate the MPC randomly across the four nodes. As shown in the figure the process is stable and is able to operate satisfactory in spite

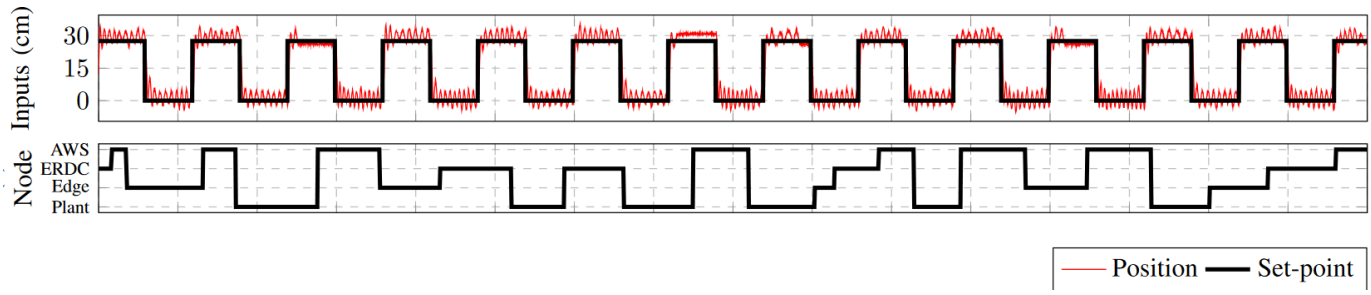


Fig. 3. MPC randomly migrated between the four nodes, from [3]. The top plot shows the setpoint in the black and the ball postion. The lower plot shows in which node the MPC executes.

of the migration and the resulting changes in latency and execution time. However, by tweaking the problem so that the optimization problem becomes more challenging it is possible to have a situation where placing the MPC in the edge node is the only viable approach. If it executes in the locally the execution times becomes too large and if it executes in the AWS the communication latency becomes too large. In both cases the ball falls of the beam. The tweaking of the problem was done by increasing the optimization horizon thus creating a larger optimization problem and by changing the reference signal to be closer to the constraints, i.e. closer to the end of the beam.

## V. Timing Issues

In the previous example the same MPC formulation and plant models were used independently of in which node it executed. This MPC formulation ignores the fact that the execution time is non-zero and time varying. However, substantially better results can be achieved if this is taken into account in the design. There are several ways of doing this and here are some examples. Dynamic variations in latency up to one sampling period can be handled by adding a one sample delay to the model. This can be achieved by using a Kalman filter that at time $t_k$ estimates what the state $x$ will be at time $t_{k+1}$. Rather than waiting with invoking the MPC until time $t_{k+1}$ the MPC is invoked immediately at time $t_k$. The resulting control signal that arrives at $t$ so that $t_k < t < t_{k+1}$ will then correspond to the control signal that should be applied at time $t_{k+1}$. By delaying the actuation of this until $t_{k+1}$ latency variations within one sample can be handled. This approach can also be extended to longer delay variations. An approach to handle the slower execution time at the local server could be to increase the sampling period of the MPC, e.g., by a factor 2, when it executes locally and resample the process model accordingly. The rationale behind this is that the sampling period selection rules used in discrete-time control typically provide a range of acceptable sampling periods rather than a single value. Another situation that must be handled is the case when the MPC is running in a cloud node and the control signal is not available at the actuation unit when needed. A possible approach for handling is this is to use the fact that a MPC returns not only the control signal to apply at $t_{k+1}$, but also the control signal to apply at $t_{k+2}$. Hence, this signal could be applied to the plant instead, effectively running the process in open loop for one additional sampling period. This is just a few of the techniques that could be used. Several more have been developed in the Networked Control System (NCS) community, e.g., [7].

## VI. Conclusions

Using the cloud for deployment of control systems has many attractive advantages. It is possible to use additional information such as high-fidelity models, additional non-local sensor data, learn from the control of other similar plants, and use the latest optimization and learning algorithm. Using this model one could think of control as a service, i.e., Control-As-A-Service (CaaS). However, the requirements on short latency requires the use of local data centers at the edge of the communication network. The migration of the controller between different nodes creates new and challenging control problems related to delay compensation, switching, and mode changes. Model-Predictive Control is a control technique that fits the cloud paradigm well.

## VII. Acknowledgements

This work has been partially supported by WASP (Wallenberg AI, Autonomous Systems and Software Program), the Lund Center for Control of Complex Engineering Systems (LCCC), and by the Swedish Research Council.

## References

[1] J. Rawlings and D. Mayne, *Model Predictive Control: Theory and Design*. Nob Hill Pub., 2009. [Online]. Available: https://books.google.se/books?id=3_rfQQAACAAJ

[2] P. Persson and O. Angelsmark, "Calvin–merging cloud and IoT," *Procedia Computer Science*, vol. 52, pp. 210–217, 2015.

[3] P. Skarin, W. Tärneberg, K.-E. Årzén, and M. Kihl, "Towards mission-critical control at the edge and over 5G," in *IEEE International Conference on Edge Computing*. IEEE Computer Society, 2018.

[4] J. Vieira, S. Malkowsky, K. Nieman, Z. Miers, N. Kundargi, L. Liu, I. Wong, V. Öwall, O. Edfors, and F. Tufvesson, "A flexible 100-antenna testbed for massive mimo," in *Globecom Workshops (GC Wkshps), 2014*. IEEE, 2014, pp. 287–293.

[5] S. Malkowsky, J. Vieira, L. Liu, P. Harris, K. Nieman, N. Kundargi, I. C. Wong, F. Tufvesson, V. Öwall, and O. Edfors, "The world's first real-time testbed for Massive MIMO: Design, implementation, and validation," *IEEE Access*, vol. 5, pp. 9073–9088, 2017.

[6] Linux Foundation, "Real-time linux," https://wiki.linuxfoundation.org/realtime/start, 2018.

[7] X. Zhang, Q. Han, and X. Yu, "Survey on recent advances in networked control systems," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1740–1752, Oct 2016.