Adaptation of Schedules and Scheduling Parameters in Cybernetic Systems

Andreas Heimrath
and Joachim Froeschl

BMW Group
R & D electronics
Munich, Germany
Email: andreas.heimrath@bmw.de

Jan Ahlbrecht
and Uwe Baumgarten

Technical University of Munich
Department of Informatics
Garching, Germany
Email: baumgaru@in.tum.de

***Abstract*—In the automotive domain, components are increasingly based on software that is being integrated through fewer and more powerful connected computing systems. With the main focus being the control of hardware with actuators and the monitoring of sensors, such systems are inherently complex. Additionally, their functions have both static and dynamic dependencies. Functions appear in a huge number with a large variety of dependencies. For decades, cybernetic approaches have been used to handle the complexity of dynamic systems in a natural manner. They have proven effective in different scientific domains like economics and management, governance, social sciences, and technical systems. Based upon the experience with a flexible energy and power management in the automotive industry (fEPM) we transfer those findings to software systems of sophisticated cars integrating more and more functions and components. We have identified Stafford Beer's Viable System Model (VSM) as a promising approach. We follow this approach step by step, starting with a basic VSM-based system/component. Using the principle of recursivity, we build horizontal and vertical combinations of systems resulting in powerful computational nodes with dynamic load depending on the individual feedback control systems and the control loops of all layers. Having the whole software stack in mind, we started to investigate modern microkernels (like seL4, Fiasco.OC and other) with their components. The scheduling, besides others, is the most promising component, where a VSM structure can be established. Here, we introduce an adaptive scheduler for the scheduling of an automotive workload, in which a machine learning process could exist. This is in line with other approaches in the field of autonomic or organic computing. A simplified energy and power management serves as an example for illustrating the applicability and usefulness of Beer's VSM in technical software systems.**

***Index Terms*—Software Cybernetics; Microkernel; Adaptive Scheduling.**

I. Introduction

In the automotive domain, components are increasingly based on software and are integrated in a few powerful connected computing systems. Nevertheless, their main focus is controlling hardware, sensors and actuators. Intelligent power and energy management for a car is a good example, where we have made encouraging experiences with cybernetics in the development and operation of a complex system.

For the purpose of controlling the main components, a cybernetic approach is used. Following the ideas of the Viable System Model (VSM) [1] the fEPM [2] was built in a successful manner.

With the growth of software-intensive components and services within a connected car, the basic software system gains importance. Therefore, we started investigating software cybernetic approaches for building the basic systems software blocks. On the one hand we analyzed software cybernetic approaches in the context of microkernel construction. On the other hand we examined the approaches of organic computing. Together with the experience of the energy and power management fEPM, we identified a VSM-based approach for constructing complex systems, that combines the best ideas of different worlds.

The rest of the paper is structured as follows. Section II will introduce the concepts of cybernetics, VSM, and microkernels, while discussing some related work. Later, in Section III we present our approach in constructing a complex (automotive) system along the ideas of VSM, putting special emphasis on adaptive scheduling. After introducing the workload to be scheduled in Section IV, we present our current solution for scheduling for a simplified energy and power management in Section V. Trying to prove the concepts presented in our current solution Section VI focuses around the creation and evaluation of a prototype and we then draw our conclusions and suggest future work in Section VII.

II. Background with related work

The application of cybernetics has a long tradition in handling complex systems. Looking at computer systems, embedded systems, internet of things (IoT) or cyber physical systems the initiatives of autonomic and organic computing have shown remarkable impact on systems architecture - both in hardware and in software - and modeling. With autonomic computing IBM developed first very fundamental steps [3] including self-configuration, self-healing and self-optimization. The paradigm shift with organic computing and further ideas and details may be found in [4]. Paradigms of biology with respect to self-organization will be applied there.

In the area of complex technical systems the cybernetic approach found its way in terms of organic computing and others.

The transition to distributed control with a strengthening of autonomy of systems and their components can be seen, for example, in [5]. A clear consequence was the development of the cyber organic system model (see [6]) with its application in the automotive domain. Further approaches taking operating systems into account may be found using the example of ubiquitous computing (see [7]). Similarly, software cybernetics occurs in the area of complex software systems. Yang et. al. (see [8]) take lots of approaches into consideration including Stafford Beer's Viable System Model.

A. Cybernetics and VSM

The science of cybernetics was founded in 1948 by the results of research of Norbert Wiener [9]. For Wiener's original work see [10]. Based around these ideas the research of Beer resulted in his VSM [11]. The VSM is an abstract model of the nervous system. It combines five subsystems to create a management model for business applications. The subsystem 1 (named as system level 1 in Figure 1) represents an operational unit. The subsystem 2 (system level 2) creates a metasystem for the coordination of all subsystems of type 1 (operational units). The function of stabilization is the main task of subsystem 3 (system level 1). The information input is given by the subsystems 4 and 5 and the metasystem of subsystem 2. The regulation function is similar to the autonomic nervous system as a cooperation of the sympathetic and parasympathetic nervous system. At least the subsystems 1 to 3 create a functionality called homeostasis. This means that all activities and procedures result in an intrinsic behavior to reach always a balanced system state or equilibrium. Beer called it "autonomics". The subsystem 4 (system level 4) combines the information of the system environment with the system information built in subsystem 3. Finally, the subsystem 5 (system level 5) has the purpose to make decisions and to give instructions to the lower instances within the model. Beer called the subsystems 3 to 5 "corporate management". Figure 1 in the next section presents the VSM solution for the fEPM.

Advancing this concept, Malik includes the principle of recursivity to couple more than one VSM to manage more extensive systems, e.g., big companies in an evolutionary way [12]. Recursivity means that an operational unit is implemented with a VSM in a similar way depending on the local tasks and their abstraction.

In addition, the VSM will be applied in various domains, like economics, governance with a sustainable equilibrium [13] and software systems (see section II-C).

B. OC, E/E, and fEPM

There are several applications of the VSM. One implementation is the flexible energy and power management as a technical approach for coordinated power supply systems. The basic model of the fEPM is shown in Figure 1. The fEPM features all five system levels [2].

SL1 Contains the system values, the physical connection control and the control functions.

SL2 Condenses the system values into operating figures and has to combine the level 1 functionalities in a fast way.

SL3 Determines the operating figures and tendencies with deposited knowledge into system states. This level contains the autonomous, state based system modifications for the purpose of system stabilization. Compared to biological systems, it works like a reflex system.

SL4 Combines the internal system states and diagnostic information from the system itself with the external system states based on environment information. The instruction of a higher hierarchy level is included here. The coupling of the environment information is made of filtered, relevant information out of the system specific environment.

SL5 Contains the operating strategy. Compared to the biological system it works like a conscious behavior. In this level the regulation values, so called modificators, are calculated.

The fEPM was originally designed for automotive systems. Another use case for VSM and its implementation for the management of stationary energy storage systems is described in [14].

From a point of view in computer science, the organic computing (OC) initiative had a bio-inspired look into aspects of systems and their implementation. A comparison of the OC-model, the fEPM-model and a combination of both called Cyber Organic System model (COS) was published by Adam et. al. in 2015 in [6]. The COS contains several layers for the basic objects, reflex, strategy, intelligence, and communication including observer/controller mechanisms. For a more detailed discussion within the topic of electric and electronic systems design E/E see [15].

A promising approach is the combination of the cybernetic modeling and artificial intelligence (see [16]).

C. VSM and Software Systems

Stafford Beer's Viable Sytem Model can be applied to software systems. The introduction as a first part in this section is based upon [17]. The second part describes our automotive example (see [18]).

Herring and Kaplan [17] are among the first scientists who applied the VSM to software with the idea of building better software systems. A complex software system, here a Viable System Architecture, integrates viable components (VC) in an overall system. Viable components are structured along the principles of VSM (see section II-A and fEPM) and provide interfaces for managing viability.

For our purpose, building a microkernel-based automotive power management, the system's characterization by Herring fits in a perfect manner. "These systems are characterized by large numbers of heterogeneous components with a high degree of interconnections, relationships and dependencies.

They exist in a dynamically changing environment that demands dynamically responding behavior. In other words, these systems must adapt to their environment."[17].

The integration of the control paradigm to (object oriented) software was postulated by Shaw [19] in a sense that a component does its own work, whereas being controlled by others. Later in section III we will use this statement for scheduling, where the creation and execution of a schedule might be adopted depending on various influences.

Corresponding to the former sections about cybernetics and VSM the principles, which will be applied to software, are autonomy, adaption, recursivity, hierarchies, invariants, and self-reference. In order to use the appropriate features and attributes of the above mentioned principles, a set of object's interfaces are introduced. These interfaces enable the objects itself, their controller and all other related components to influence each other.

The second part of this section gives an example, how the software system for automotive power management can profit from machine learning techniques, here in terms of reinforcement learning. The architecture of the VSM and of the fEPM as its extension paved the way for the development of reflex-augmented reinforcement learning (RARL) [16]. RARL extends the concept of standard reinforcement learning by integrating the biologically-inspired reflex of the fEPMs system level 3 into the learning process. This makes reinforcement learning (RL) accessible to safety-critical applications such as automotive electrical energy management. The reflex ensures that only safe actions suggested by the agent of RL are executed and efficient training as well as operating the system are guaranteed in a defined range of safe system states. In automotive energy management for example, the reflex can reject actions that could lead towards states of undervoltage causing a severe breakdown of the cars steering assistance system. RARL based on deep Q-learning fulfilled major requirements of a real automotive electrical energy management system in a vast simulation study [18]. Furthermore, RARL suggests a way to fuse RL and cybernetic management systems. This makes it possible to realize structured learning management systems even for future systems of very high complexity.

D. Microkernels

Even in the automotive domain hardware/software systems are becoming increasingly connected, software-defined and complex. Therefore, the demand for microkernels undergoes a revival. Their philosophy is (a) to try and keep the kernel as small and lightweight as possible, (b) execute almost all code in user space / user mode, and (c) establish a tiny trusted code base for secure, reliable, and resilient systems. Microkernels like L4, seL4 [20], Fiasco.OC [21] or Minix 3 (see [22]) have a long history, but new kernels, like Zircon [23], show up on the horizon. They all have in common similar function blocks, like basics for scheduling, inter process communication (IPC), the basics for security, and some basic memory or address space management. Further aspects and performance evaluation for communication may be found in [24]. With respect to cybernetics, scheduling is most appropriate and many contributions (in particular from organic computing, MAPE cycle (Monitor/Analyze/Plan/Execute from [3]) can be found in this direction. Security has some closeness, but IPC is far away from cybernetics, because of the huge requirements for performance. Some examples, where cybernetic approaches appear, are self-management in Minix 3 [22] or Fiasco.OC [21]. The monitoring of drivers in Minix 3 can discover, that a driver is inhibited in his processing and the scheduler may react on this observation. Like the MAPE cycle the scheduler in Fiasco.OC can monitor the execution of tasks and may change the strategy depending on the analysis, planning and execution steps.

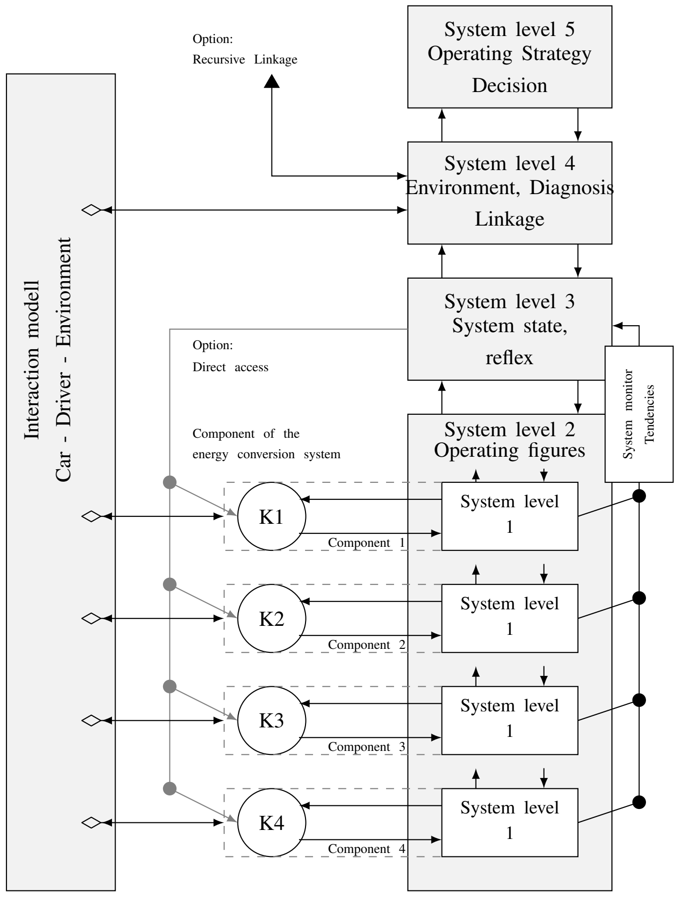

Fig. 1. fEPM.

III. Approach

In the former sections, we explained the origins and ideas of VSM and shown how VSM is used within an automotive energy and power management and how it is applied to software. The next step will be its application in the area of adaptive scheduling for the energy and power management system. This example illustrates the general approach.

The five layers of the cyber organic system model (see [6]) are adopted with respect of our scheduling approach. Starting from bottom to top the object layer includes the real (hardware) components of the energy system (generator of energy, storage for energy, and its consumers including sensors and actuators) and the necessary software components (workload and basics for scheduling/dispatching). The layer above is the reflex layer which includes first and fast analysis of the state of the real components (some of the processes in Table I). The strategy layer is responsible for the execution of the schedule by the dispatcher and the shortterm learning for the scheduler. The intelligence layer tries to learn the long term operating strategy, which is at the moment not included in our prototype. The communication layer includes the interplay with the environment (again one of the processes in Table I). For our scheduling approach, the given workload has to be scheduled and this is done in an adaptive manner. The structure of the COS shows the way to implement the scheduling. The MAPE cycle is applied and includes an appropriate learning strategy. Details including the workload are presented in chaper IV.

IV. Model of the workload and its parameters

The scheduler should operate in a non-interactive, mixed criticality environment. This means that it consists of processes that have strict check-up intervals every couple of milliseconds, which we will refer to as processes with a deadline, and other processes without deadlines. However, these check-up intervals are not strict, i.e., not a hard deadline, in the sense that if processes are missed their calculations become obsolete. Therefore, processes continue their calculation even if they miss their check-up time. Additionally, during the normal execution of the schedule, we want to train a neural network every couple of cycles, where each cycle is one execution of all our periodic processes. The neural network is itself a process and is implemented in OpenNN version 3.1. The network has two hidden layers with 25 neurons and uses simple sigmoid activations. When the process gets to run, it trains for around half a second and tries to approximate a 28 dimensional polynomial. This is done since, currently, we do not require any kind of output from the network and simply want a realistic load emulation.

The processes themselves perform operations ranging from very simple arithmetic operations like subtractions, over read out sensor data or to check if values are within a certain range, to more time intensive tasks like higher dimension matrix multiplications. The exact processes that make up the workload to be scheduled can be seen in Table I and are inspired by the tasks the fEPM has to full-fill. The whole workload consists of lots of these process types with a fixed distribution. The distribution is a based on expert knowledge, examples for possible values are 20% for Model Calculation and 25% for Risk Assessment.

TABLE I. THE PROCESSES THAT ARE PART OF THE WORKLOAD. PRIORITIES MAY BE VARIABLE (VAR) OR FIXED (FIX). THE MAXIMAL PRIORITY IS NAMED AS mpr.

Process	Check-Up Interval	Priority	Calculation
Fault Management	Yes	VAR and $\geq \frac{mpr}{2}$	$\mathbb{R}^{5\times5} \times \mathbb{R}^{5\times5}$
Range Check	Yes	VAR and $\geq \frac{mpr}{2}$	$\mathbb{R} - \mathbb{R}$
Risk Assessment	No	VAR and $< \frac{mpr}{2}$	$\mathbb{N} - \mathbb{N}$
Error Calculation	Yes	FIX and $\geq \frac{mpr}{2}$	$\mathbb{R} - \mathbb{R}$
Functional Safety	No	FIX and $< \frac{mpr}{2}$	$\mathbb{R}^{100\times100} \times \mathbb{R}^{100\times100}$
Model Calculation	Yes	FIX and $\geq \frac{mpr}{2}$	$\mathbb{R}^{30\times30} \times \mathbb{R}^{30\times30}$
Training a Neural Network	No	FIX at $\frac{mpr}{4}$	$\mathbb{R}^{28} \to \mathbb{R}$
Using a Neural Network	No	FIX and $< \frac{mpr}{2}$	$\mathbb{R}^{28} \to \mathbb{R}$

V. Solution

In this section we describe the scheduling aspects and our proposed scheduler.

A. Scheduling Method

There were two real options we considered for scheduling approaches. On the one hand, a multi-level priority scheduler [25], which assumes that two processes are not equally important and assigns them a priority. It then orders processes by their priority and, starting with the highest priority, starts to dispatch processes within the same priority using another scheduling method, for example, round-robin.

On the other hand, a batch scheduler [25] that just takes processes that are ready and packages them into a batch. This batch represents the plan that is then handed to the dispatcher and is executed one after another, without interruptions.

Our scheduler needs to able to differentiate between more or less important processes, based on their criticality. But, our domain (i.e. automotive) still consists of a large subset of processes with predictably static behaviour. Therefore, we decided to go for a mix between a multilevel and a batch scheduler.

B. The Proposed Scheduler

The scheduler tries to automatically adapt parameters to keep process execution within check-up intervals. It schedules in batches using MAPE [3] like cycles. After monitoring some parameters such as the amount of check-up intervals missed after every execution of a batch, the performance during the last cycle is evaluated and, if needed, adaptations are made for the next scheduling cycle.

Despite the usual inflexibility of a batch scheduler, it fits our needs since there is no outside interaction necessary and the processes calculations are short enough that the scheduler does not have to be preemptive. However, certain processes, like the one responsible for the training of the neural network, still have hard to predict run times. Therefore, our scheduler requires the ability to differentiate the importance of processes, to create a good batch. As a result, we assign a priority variable to processes, allowing us to influence the kind of batches our scheduler creates and to combat some of the downsides that come with batch scheduling.

Currently, there are two main adaptations our scheduler can make to improve performance in either the amount of check-up intervals held or fair service. One of them we call the minimum priority barrier and the other priority boosts. The first, as the name suggests, attempts to handle higher load situations by simply increasing the minimum priority required to be added to the next batch. Higher load situations are detected by monitoring the amount of check-up intervals kept. If a certain number of check-ups happens too late, the scheduler increases the minimum priority to avoid further check-up intervals being missed, by cutting out non-critical processes.

With the introduction of a system that cuts less important processes, there is the risk of starvation [26]. Starvation is when less important processes do not get to run for longer time periods. If the minimum priority has been increased, the scheduler offers two different ways of avoiding starvation:

The first is for the minimum priority to be decreased again, this happens if check-up intervals are being kept. The scheduler would then decrease the minimum priority and lower priority processes would get to run again.

The second relies on another variable that is being kept by each process called hunger. It determines how many cycles, so batch creations and dispatches, in a row a process had to be excluded from the batch despite being ready. If this variable reaches a certain threshold the priority of the process gets a one time only increase. In our current implementation this means it will have its priority increased to the minimum priority necessary to be included in the next batch. Other versions of the priority increase, like always increasing it by a fixed amount, are also possible. This way we avoid starving lower priority processes while still staying focused on keeping check-up intervals.

The scheduling approach proposed has been inspired by the Fiasco.OC microkernels scheduling approach. Fiasco.OC uses a so called context [27] to store parameters that can be monitored for MAPE like cycles. We also created such a context for our processes to make parameters easy to monitor.

VI. Evaluation with our Prototype

As a proof of concept, we built a prototype in C++ that emulates the execution of the processes controlled by the proposed scheduler and a dispatcher. The current implementation is not directly directly on hardware and therefore suffers from overhead through the underlying operating system (Ubuntu 18.04). However, we deemed this sufficient to determine the feasibility of our approach before an actual implementation.

The processes that make up the workload are deployed by pthreads. The dispatcher uses locks for their coordination. Each process is represented by a pthread that executes an

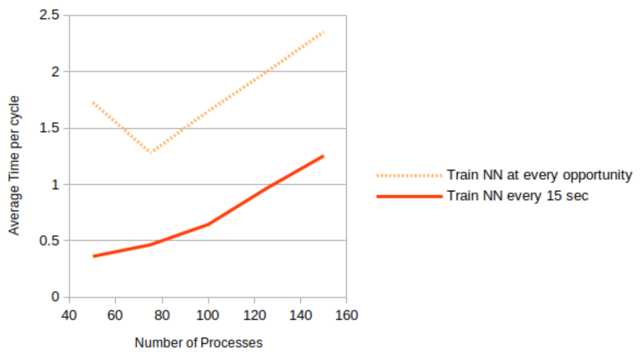

Fig. 2. Variation between neural network training intervals and their effect on the average time per cycle.

operation which is similar to the computation the components would execute. We did not model the physical components, but only the processes that approximate their behaviour in the real world. Dispatching is simulated through locks, that get unlocked if a process has been assigned calculation time. The dispatcher holds all the locks in the beginning and then unlocks a certain number of locks at the same time to enable processes to run parallel to each other. In our prototype we assumed that a fixed number of threads could run simultaneously. The plan how to dispatch the processes is created by the scheduler as mentioned before.

We then ran a couple of tests with a different number of total processes while always keeping a singular process responsible for training the neural network and one for using it. First, we wanted to variate some of the parameters of the workload to see the impact on the time per cycle. We started of by testing how the training intervals of the neural network affected the scheduler’s performance. For this purpose, we ran two tests, one where we tried to train the neural network as often as the minimum priority barrier allowed, i.e., at every opportunity, and another one where we only scheduled the neural network less frequently, here every fifteen seconds. After executing the processes for 100 cycles for 20 experiments, we looked at the average time it took for our scheduler to complete one cycle. The results can be seen in Figure 2. As we can see, training the neural network in intervals instead of constantly results in more than half the time per cycle and therefore better scheduling performance. The dip in time per cycles for constant training at around 75 processes is due to the minimum priority barrier and will be explained later on.

Next, we fixed the neural network training interval to 15 seconds and changed the percentage of functional safety processes in the total number of processes. We then once again ran for 100 cycles for 20 times and observed the time taken per cycle. While there was a visible difference of around 0.1 seconds for each cycle (see Figure 3), the impact of changing the neural network training interval was a lot greater.

We also wanted to observe the effect the minimum priority barrier had on the percentage of check-up intervals missed.

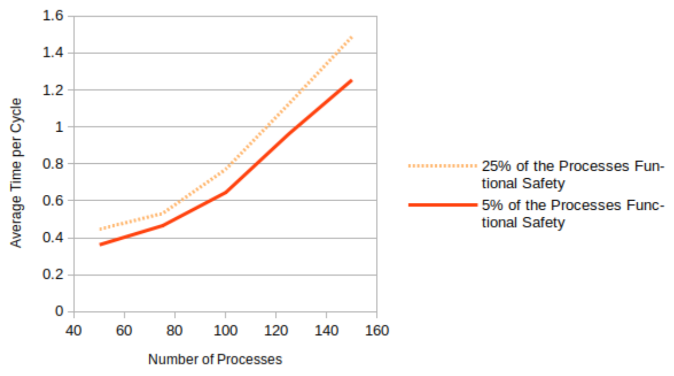

Fig. 3. Impact of the amount of functional safety process on the average time per cycle.

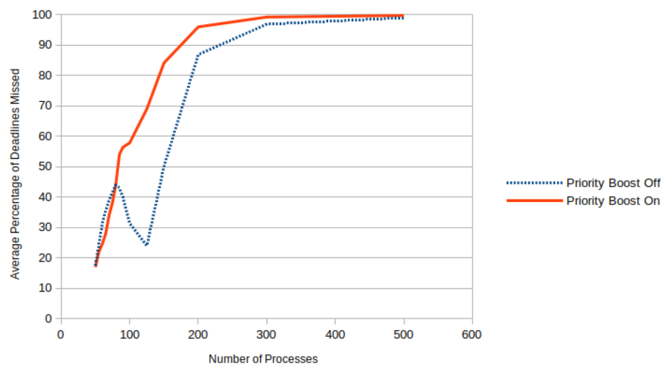

Fig. 4. The graph shows the average percentage of missed check-up intervals per cycle for a certain number of processes.

Figure 4 shows the relationship between the percentage of check-up intervals missed and the number of processes that are currently being scheduled. As to be expected, with an increase in the number of processes the scheduler eventually overloads and the amount of check-up intervals missed increases dramatically. However, before that occurs, at around 80 to 125 processes the amount of check-up intervals missed decreases again. This can be explained through the minimum priority barrier taking effect. It reacts to the higher load and starts cutting processes to maintain the check-up intervals. The reason the amount of check-up intervals missed drops so drastically is due to how the workload is made up. Only less critical processes without check-up intervals can be cut, which in our workload are the processes that require the most time to execute. During an increase in the minimum priority required, more and more of the processes without check-up intervals get cut until eventually, at around 125 processes, the minimum priority barrier is constantly maximized, meaning that all processes that could have been excluded from being scheduled, have been excluded. Now, with an increase in the total number of processes the percentage of check-up intervals missed increases again.

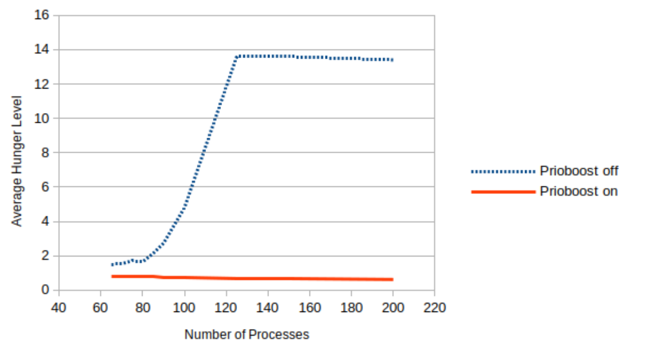

Fig. 5. The graph shows the average hunger level of the processes for a certain number of processes.

Next we wanted to see if the priority boosts had the intended effect of keeping a fair service to all processes. While Figure 4 might give the impression that having the priority boost enabled is solely a disadvantage, a look a Figure 5 shows otherwise. Figure 5 displays a close up of the average value of the hunger variable described earlier, in the area where the dip in the percentage of check-up intervals missed happened in Figure 4. From the graph we can observe a drastic increase in the hunger variable around that time, meaning that processes without check-up intervals are no longer being assigned processor time, essentially starving them. Meanwhile, the curve with priority boosts enabled keeps the hunger variable at a constant level showing that priority boosts can be used to ensure fair service.

VII. Conclusion and Future Work

Concluding, we have shown that methods such as the minimum priority barrier and priority boosts have the intended effect of making the scheduler adaptive to changes in the load, while being able to maintain the intended goals of keeping check-up intervals, fair service or a mix of both.

Due to its adaptability our scheduling approach enables the design of complex systems without previous extensive knowledge of process parameters. This, while not providing the error rate needed for safety-critical systems, paves the way for a wide variety of applications not limited to automotive energy management to keep soft deadlines without any previous knowledge of the run time of other tasks in the system.

In the future, it would be interesting to experiment with a reinforcement learning approach to set parameters such as the one for the priority barrier dynamically based on the current processes ready for scheduling. This could lead to even more adaptable scheduler.

Currently, the scheduler struggles due to its non-native performance and the resulting increase in time needed for a context switch. Implementing a prototype of the scheduler in an operating system and testing it using native performance would definitely be among the next steps.

Additionally, further tuning of the parameters for the minimum priority barrier and priority boosts is necessary to better serve the current workload. Furthermore, the scheduler could be extended to be able to tell if it is barely able keep check-up intervals and as a result to avoid unnecessary decreases in the minimum priority. But, a lower-level implementation with native performance is required for any kind of fine tuning, otherwise cycles are too unreliable.

REFERENCES

[1] S. Beer, *Cybernetics and Management*. Wiley, 1959.

[2] J. Fröschl, S. Kurtz, M. Winter, J. Taube, T. Nuritdinow, and H.-G. Herzog, "Concept of a decision system for an operating strategy in a cybernetic energy and power management," in *EEHE2016*, 2016.

[3] IBM, "An architectural blueprint for autonomic computing," https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf (Visited on 28.02.2020), IBM, Tech. Rep., 2005.

[4] C. Müller-Schloer, H. Schmeck, and T. Ungerer, *Organic Computing A Paradigm Shift for Complex Systems*. Basel: Birkhäuser, 2011.

[5] S. Tomforde, B. Sick, and C. Müller-Schloer, "Organic computing in the spotlight," *arXiv:1701.08125 [cs.MA]*, 2017.

[6] D. Adam, J. Froeschl, U. Baumgarten, A. Herkersdorf, and H.-G. Herzog, "Cyber Organic System-Model New Approach for Automotive System Design," in *ADAPTIVE 2015:The Seventh International Conference on Adaptive and Self-Adaptive Systems and Applications*, 2015, pp. 92–97.

[7] M. Mattos, "Organic computing and operating systems: The big challenge," *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, pp. 392–1397, 2015.

[8] H. Yang, F. Chen, and S. Aliyu, "Modern Software Cybernetics: New Trends," *Journal of Systems and Software*, vol. 124, pp. 169–186, 2017.

[9] N. Wiener, *Kybernetik*. Duesseldorf: Econ, 2. Auflage, 1963.

[10] ——, *Cybernetics*. New York: John Wiley u. Sons, 1948.

[11] S. Beer, *Kybernetik und Management*. dt. Ausgabe, Fischer Verlag, 1962.

[12] F. Malik, *Strategie des Managements komplexer Systeme*. Haupt Verlag, 9. Auflage, 2006.

[13] S. Barile, B. Quattrociocchi, M. Calabrese, and F. Iandolo, "Sustainability and the viable systems approach: Opportunities and issues for the governance of the territory," *Sustainability*, vol. 10, pp. 1–17, 2018.

[14] H.-G. Herzog and et. al., "Applications of the viable system model in automotive and battery storage systems," in *2016 IEEE International Conference on Systems, Man, and Cybernetics; SMC 2016 — October 9-12, 2016; Budapest, Hungary*, 2016, pp. 1714–1752.

[15] D. Adam, *Concept of bionic E/E architecture for future vehicles based on the model of the human body (in German)*. Dissertation at TUM, 2016.

[16] A. Heimrath, J. Froeschl, and U. Baumgarten, "Reflex-augmented reinforcement learning for electrical energy management in vehicles," in *Proceedings of the 2018 International Conference on Artificial Intelligence, CSREA Press*, 2018, pp. 419–430.

[17] C. Herring and S. Kaplan, "The Viable System Model for Software," *4th World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000)*, 2000.

[18] A. Heimrath, J. Froeschl, R. Rezaei, M. Lamprecht, and U. Baumgarten, "Reflex-augmented reinforcement learning for operating strategies in automotive electrical energy management," in *Proceedings of the 2019 International Conference on Computing, Electronics & Communications Engineering (iCCECE), IEEE*, 2019, pp. 62–67.

[19] M. Shaw, "Beyond objects: a software design paradigm based on process control," *ACM SIGSOFT Software Engineering Notes*, vol. 20, no. 1, pp. 27–38, 1995.

[20] G. Klein and et. al., "sel4: Formal verification of an os kernel," in *Proceedings of the ACM 22nd Symposium on Operating Systems Principles, ACM*, 2009, pp. 207–220.

[21] FIASCO, "The fiasco.oc microkernel," http://os.inf.tu-dresden.de/fiasco/ (Visited on28.02.2020).

[22] A. S. Tanenbaum, "Lessons learned from 30 years of minix," *Communications of the ACM*, vol. 59, p. 3, 2016.

[23] FUCHSIA, "The zircon microkernel," https://fuchsia.dev/fuchsia-src/concepts/kernel (Visited on28.02.2020).

[24] Z. Mi, D. Li, Z. Yang, X. Wang, and H. Chen, "Skybridge: Fast and secure inter-process communication for microkernels," in *EuroSys 19, March 2528, 2019, Dresden, Germany*, 2019.

[25] A. S. Tanenbaum, *Operating Systems Design and Implementations*. Pearson, 2006.

[26] A. S. Tannenbaum, *Modern Operating Systems*. Prentice Hall, 2001.

[27] D. A. Krefft, "Flexible task management for self-adaptation of mixed-criticality systems with an automotive example," Ph.D. dissertation, Technische Universitaet Muenchen, 22.08.2018.