RTAS 2013

19th IEEE Real-Time and Embedded Technology and Applications Symposium

Philadelphia, USA
April 9 - 11, 2013

Work-in-Progress Proceedings

Edited by Marko Bertogna

Message from the Work-in-Progress Chair

Welcome to the RTAS 2013 Work-in-Progress (WiP) session. This session is dedicated to new and ongoing research in real-time systems, theory, and applications. The papers included in these proceedings cover a wide range of topics spanning across all areas of real-time and embedded systems, including multi-core processors, energy and power-aware systems, distributed systems, automotive architectures, and wireless sensor networks.

The primary purpose of this WiP session is to provide researchers with an opportunity to discuss their evolving ideas and gather feedback from the real-time systems community at large. Out of the 18 submissions that were received in response to our call for papers, the program committee selected 12 papers, which appear in these proceedings. I would like to thank the members of the WiP technical program committee for their help, and all the authors for submitting their work to this forum. I am sure that many of the papers featuring in these proceedings will appear as full-fledged conference and journal papers in the near future.

I hope that all of you enjoy this session, participate in stimulating discussions, and provide the authors useful feedback concerning their work.

Marko Bertogna
Real-Time and Algorithmic Research Group
University of Modena, Italy
RTAS 2013 WiP Chair

RTAS 2013 Work-in-Progress Technical Program Committee

WORK-IN-PROGRESS CHAIR

Marko Bertogna, University of Modena, Italy

PROGRAM COMMITTEE

- Benny Åkesson, Eindhoven University of Technology, Netherlands
- Moris Behnam, Mälardalen University, Sweden
- Konstantinos Bletsas, Polytechnic Institute of Porto, Portugal
- Björn B. Brandenburg, Max Planck Institute for Software Systems, Germany
- Tommaso Cucinotta, Bell Laboratories Alcatel-Lucent, Ireland
- Jeremy Erickson, University of North Carolina at Chapel Hill, USA
- Nathan Fisher, Wayne State University, USA
- Shinpei Kato, Nagoya University, Japan
- Jinkyu Lee, University of Michigan, USA
- Patricia López Martinez, University of Cantabria, Spain
- Mauro Marinoni, Scuola Superiore Sant'Anna, Italy
- Ahlem Mifdaoui, University of Toulouse/ISAE, France
- Claire Pagetti, ONERA, France
- Rodolfo Pellizzoni, University of Waterloo, Canada
- Harini Ramaprasad, Southern Illinois University Carbondale, USA

Table of Contents

A Service Oriented Smart Home Architecture for Better Quality of Service Management

Can Basaran*, Homin Park*, Taejoon Park*, Sang H. Son*
*Department of Information and Communication Engineering, DGIST, Daegu, Korea
{cbasaran, andrewpark, tjpark, son}@dgist.ac.kr

***Abstract*—Smart home environments require tight coupling between different types of sensors, actuators, and computer algorithms. To address the challenges in building a robust and effective smart home environment, it is essential to develop a systematic approach to handling a large number of components and their interactions. We describe a holistic, multi-layered, and service oriented approach to smart home environment design that will enable abstractions needed for enforcing performance constraints. This paper presents our approach and presents preliminary results with a Quality of Service manager that enforces load balancing. We show that a simple load balancing scheme can be very effective compared to intuitive ad-hoc implementations of representative tasks.**

I. Introduction

There has been extensive research on smart home services focusing on specific applications such as activity recognition/automation, energy conservation, early warning systems, and health care systems [1], [2]. In contrast, the previous work on hosting platforms for smart home applications that can enforce quality of service (QoS) constraints is relatively low in number [3], [4]. However, the ability to enforce constraints on performance metrics, e.g., response time, availability, and sensing accuracy, is necessary for implementation of mission critical applications. Embedding ad-hoc algorithms in order to enforce necessary constraints in services and applications that require them is challenging. First of all, sensing hardware as well as low level services such as identification, and localization are shared between higher level applications and these shared resources demand global QoS management. Secondly, ad-hoc implementations decrease robustness by increasing the amount of repeated code and hardware, which, in turn, increases the run-time overhead of the overall system. A system wide QoS manager can make smarter decisions more efficiently and has the possibility to enforce policies considering relative importance of smart home applications running in the same environment.

Typically, smart home implementations follow a centralized architecture where the sensor data is collected at a server and actuator signals are generated from the same server. However, applications that require timely processing of high rate sensor data streams are more suited to distributed architectures especially when they coexist with other applications. For example, fall detection systems based on depth cameras process around 10 megabytes of data per second [5] and in the existence of a large number of depth cameras, a distributed architecture may increase throughput. At least hardware and software discovery, data storage and retrieval, distributed stream data processing, communications, and QoS services are needed for applications and services that collectively utilize the shared sensors and actuators. In this paper, we describe a component based, multi-layered, service oriented software stack that can enforce QoS constraints and provide system level services for smart home applications. The system consists of five layers: the application layer, the system service layer, the data services layer, the network layer, and the physical layer. This architecture is being implemented for the DGIST Smart Home, a new smart home environment within the DGIST campus.

In Section II we describe the components of the application layer and various application scenarios. Section III explains the system level services provided for the application layer. In Sections IV and V, we describe the data services and the network layers respectively. Section VI briefly explains the sensor and actuator hardware necessary for a smart home. In Section VII we evaluate an initial version of our system by implementing load balancing and present the results. Section VIII gives a brief summary of our efforts and concludes the paper.

II. Application Layer

In this section we define a number of applications and services to be implemented for a smart home. We also determine technical requirements and services necessary for the implementation of these applications in a real environment. These requirements are given in terms of system level services, physical sensors, and actuators.

Harm Prediction Engine (HPE) generates alert messages in the system when projected harm to an object is detected [6], [7]. We define an object to be any sensible entity in the environment including inhabitants and appliances.

Smart Government Terminal (SGT) enables citizens to interact with the government from their homes. Since the smart home is a proof of residence, and houses different types of sensors for authentication purposes, it is an effective environment to deploy an SGT. Users are authenticated in the environment by means of finger-printing, face, and retina recognition techniques. Once a user is authenticated, the identity is attached to the user and tracked along with her. It is possible to cast a ballot through an SGT, or to print out legal documents. Furthermore, SGT can provide data to the government through a secure connection, making it possible to generate census reports as a response to flexible queries.

Virtual Smart Gadgets (VSG) system supports third parties to implement cyber-physical gadgets for the smart home. These virtual gadgets can be through body or hand gestures and

can respond to touch just like physical tools. For example, a calendar or a clock projected on the wall that can be touch-controlled for customization.

Personal Information Flow (PIF) provides a continuous flow of information to each individual inhabitant. The information stream contains e-mail messages, updates from social networking sites, financial data, and other personal subscriptions. The updates are presented on available displays, projected on to objects, or announced through speakers. To determine the feed presentation medium, the system receives attention tracking information for each individual in the environment.

Personal Resource Metering (PRM) application tracks consumption of different resources per-inhabitant, per-appliance enabling fair billing and personal tariffs.

Smart Ambiance Control (SAC) application controls the ambiance lighting and climate based on personal preferences and various other inputs such as time of day, or ongoing activity.

Health Care Service (HCS) tracks the health status of residents. Information gathered from various medical or multi-purpose sensors such as cameras, microphones, motion trackers, as well as inferred knowledge gathered from lower layer services are continuously analyzed and stored. In emergency cases a warning message is sent to health care professionals.

Activity Automation Service (AAS) automates activities based on historical data on inhabitant behavior using a feedback mechanism to continuously refine and update activation rules.

III. System Services Layer

This section describes the system level services provided in a smart home. These services are used by the applications and higher level services in the application layer. All services in the environment provide a publish-subscribe interface which allows service-to-service and service-to-application communications.

Object Identification Service (OID) is responsible for determining the identities of objects. These objects can be humans, appliances, or any material object that is inside the smart home environment. Identification includes categorization of objects under generic names such as human, milk, or telephone. Furthermore, determining the actual identity of the human, or referring to a specific milk bottle falls into this domain.

Object Tracking Service (OTS) keeps track of physical locations of objects in the smart home. This service gathers object identities from the OID service and updates location information for each mobile object.

Attention Tracking (ATR) is responsible for creating a list of objects that are, consciously or unconsciously, of interest to an entity at a given time. This list is called a *set of interest*. The entity can be a human, an animal, or an artificial intelligence, such as a robot. ATR creates a list of subjects with associated interest confidences and historical weights. Confidences describe the predicted level of interest. Historical weights are used to *forget* items from the list. A subject's historical weight decreases over time unless the interaction with that subject or related subjects continues. An object is said to be related to another object if these objects frequently fall into the same set of interest.

Context Identification (CID) service extracts relations from outputs of other system level services, such as ATR, and sensors in the smart home. A *context* is a set of related sensor readings and service outputs, in which, the relation is the intent of an entity or a cause and effect relation. CID tries to bundle together different sensor readings, one of which is time, into this relational set. Each sensor reading has a degree of membership to a context.

Activity Identification (AID) service generates textual labels for contexts to make interaction with outside world possible. These textual labels are necessary, for example, to create audio feedback messages, send textual messages, to do look-ups in search engines, and to put purchasing orders to online stores.

IV. Data Services Layer

Data services layer provides data storage and processing services. This layer, together with the network layer below, abstracts out the underlying distributed system and creates an illusion of a single machine.

QoS Manager enforces priority, response time, and other QoS policies by monitoring and altering the state of various hardware, such as sensors/actuators, as well as software services [8]. Each room has a number of relatively powerful processing elements (PEs) that are capable of processing and storing data. Local QoS managers keep track of load levels on each PE and work cooperatively in order to enforce priority levels requested by smart applications. When the load on a PE exceeds a certain threshold, QoS manager in that node finds an idle PE to migrate some of the load. Migration candidates are selected according to their priorities. Lowest priority tasks are migrated first. A lower level service, such as the CID service, inherits the priority of the highest priority subscriber. For example, if the health care system is using the CID service, then the priority of the CID service is at least as high as the health care system.

Distributed File and Database System (DFDBS) provides a single name-space across the PEs and data replication services in order to increase the robustness and availability of the system [9]. A distributed lightweight database service stores sensor readings and information generated by smart services. Sensor data streams are published/subscribed by an SQL like syntax and underlying storage facilities are not visible to the layers above this layer. Instead of addressing individual PEs, or sensors/actuators, smart applications use semantic addressing. For example, "*read data from all motion sensors inside the living room*", "*log data from the camera that is closest to an active motion sensor.*", "*if a photo sensor reading exceeds some value, turn off the lights in that room*".

Data Stream Processing System (DSMS) allows applications and services to process continuous sensor data distributively with a small amount of application specific code. Services and applications are composed of modules, i.e., functions with a well defined interface, that can be migrated or replicated. Inputs and outputs of a module have the same structure: a pointer to a data buffer, and the size of the data. These modules are linked together as a pipeline by feeding the output of a preceding module into the input of the following module. The order of execution is explicitly specified by the applications. DSMS system is responsible for passing data streams between

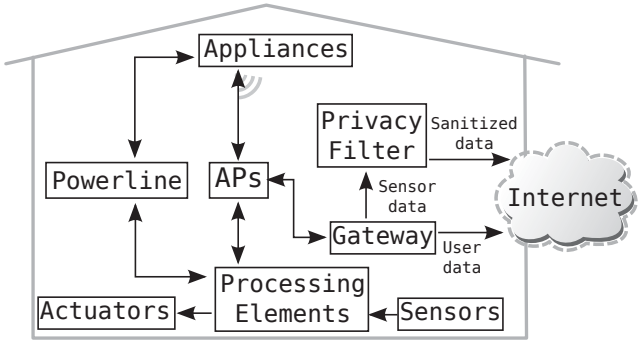


Fig. 1. Network layer data flow diagram

modules and ensuring a correct execution order between them. DSMS prefers the main memory of a PE for data processing, and switches to the secondary storage only if the main memory is not sufficient for current tasks.

V. Network Layer

The network layer enables connectivity between individual services, sensors, and actuators deployed within the smart home as well as to entities outside the environment.

As shown in Figure 1, a smart home network consists of wired and wireless communication channels. The wireless data is routed by the access points distributed around the smart home. These access points have an additional Ethernet interface that connects them to the gateway, which in turn is connected to the Internet. The gateway can differentiate between smart home generated network packets and packets transmitted by users. Smart home generated packets are first sent to the *privacy module* for inspection and *sanitization*.

Internal connectivity between services, applications, sensors, and actuators is also handled by the network layer. There is one-to-one relationship between PEs and sensors/actuators and at most one TCP connection between each PE pair. PEs continuously announce themselves through UDP broadcast messages. A broadcast message contains physical location of the PE and the types of sensors/actuators controlled by the PE. These broadcasts are handled by the discovery module that compiles a catalog of available hardware and services in the smart environment. When a local application or service submits a query, query destinations are first resolved by this discovery module, hence, above the network layer all addressing is done by sensor types and locations and the smart environment looks like a single system.

Transport module is responsible for creating and terminating connections between PEs. When a query requires a connection to a remote PE, transport module checks the list of connections and only creates a connection if currently there is no established connection to the destination PE. When data from a remote PE arrives at the node, transport module looks at the source sensor type, sensor location, and sensor identifier. The incoming data is then forwarded to the local subscribers via shared memory granting read-only access.

VI. Physical Layer

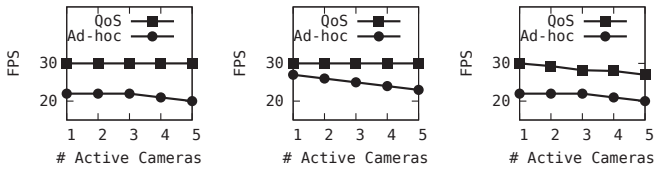
Physical layer consists of sensors and actuators deployed in the environment as well as the communication infrastructure described in Section V.

Contact sensors sense the states of lids, doors, and windows. *Pressure sensors* measure the pressure on surfaces. Floors, table tops, and surfaces contain a number of pressure sensors to detect presence of objects. *Motion sensors* category contains passive infrared, ultrasonic, microwave, and tomographic sensors to be used for activation of more complex systems in presence of movement. *Cameras* are used for detection of motion, recognition of objects, and facial features. *RFID systems*, i.e., RFID tags and readers are used for object identification and tracking. Smart home friendly household goods, and food products contain RFID tags to be are queried in online databases for textual labels, warning codes, production dates, and cooking instructions. *Microphones* and microphone arrays are used for voice recognition systems, detecting the level of background noise, and in applications that require audio processing. *Temperature sensors* measure the ambiance temperature. They are utilized by a number of applications. For example, CID uses temperature readings as a part of contexts when there is sufficient correlation between ambient temperature and other readings. *Depth cameras* provide depth information for object detection and recognition [10]. Depth cameras facilitate in object recognition while also providing movement and night vision. Smart home houses a number of depth cameras in each room. Depth cameras are connected to the environment through PEs. *Resource meters* are used for monitoring consumption of resources such as electricity, gas, water, and network bandwidth, as well as resource leaks in the environment.

Smart home houses a number of actuators for controlling the environment and communication. *Displays and projectors* show textual messages, videos, or images to users. The smart home user interface is accessible from any screen or suitable surface in the environment. *Speakers* are used for generating audio messages such as speech, notification, and alert sounds. *Resource storage systems* provide means to store resources for future use. The resources that can be stored are electricity, water, and solar energy [11]. Smart home AI can initiate the resource storage process or switch from main-line input to the storage depending on transitive resource costs and consumption behavior. *Automatic blinds* change the ambiance lighting depending on the current context and personal preferences. *Appliances* can be used for changing the ambiance, or for many other application defined purposes such as energy saving. *Lighting control systems* allow smart home applications and services to control all of the light sources in the environment. Individual lights can be dimmed, turned on/off.

VII. Preliminary Results

We evaluate the prototype implementation of our system in the DGIST Smart Home test-bed that covers and area of 100 square meters and contains a bedroom, a study room, a kitchen/dining room, a multi-media room, and a bathroom. Each room contains 4 depth cameras connected to a separate PE. PEs are mini PCs equipped with a dual-core 2.93 GHz processor, 4GB of RAM, and 300GB of persistent storage running Linux kernel version 3.7.7-200. Depth, joint, and RGB data are collected distributively by these PEs. The test-bed hosts 40 photo sensors, 40 contact sensors, 10 wireless window and door sensors, and a large number of pressure



(a) OID (b) HCS (c) GUI

Fig. 2. Performance of different applications with and without QoS Manager

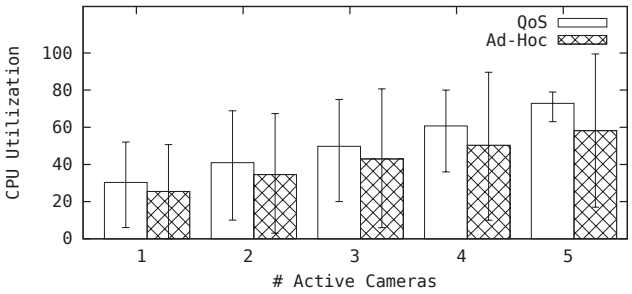


Fig. 3. CPU utilizations across all PEs with and without QoS Manager

sensors. Each room contains two high definition cameras, and 2 microphones that are connected to the PEs through an 802.11 connection. Other sensors include wireless ECG body sensors, sleep monitoring devices, activity trackers, RFID tags, RFID readers, display devices, and speakers.

An initial prototype of the described system is implemented on the DGIST Smart Home in order to evaluate the QoS manager module. We have used three tasks for evaluation. *OID* task represents the object identification service and implemented as a Support Vector Machine based Hog person detection algorithm over the camera data [12]. *HCS* task represents the health care service. The input to this task is randomly generated positions for 20 joints on the human body. These positions are updated 30 times per second and two second windows are compared against 50 predefined displacement vectors of 20 joints, that are assumed to define critical conditions. The *GUI* task is a user interface that displays camera views from up to 10 cameras on the same screen. For these preliminary experiments inputs to OID and HCS tasks are generated randomly. The GUI task updates the screen as fast as possible as it receives new data from 10 different cameras. We measure the performance of the task set in terms of the number of data frames processed per second. We compare the QoS Manager, which currently only implements CPU load balancing, against ad-hoc implementations of the tasks distributed intuitively to process data at the data sources in order to increase parallelism. When executed as a single application, OID, HCS, and GUI generate 27%, 23%, and 46% CPU load on each PE, respectively. Only 10 PEs and depth cameras were active during the evaluation.

Figure 2 shows the data processing rate as an average of 10 runs per application, when all applications run concurrently. The x axis shows the number of cameras actively generating data and the y axis shows the number of frames processed by an application. We observe that the ad-hoc implementations overload the source nodes and none of the applications can process the complete sensor data each second. The QoS manager utilizes idle nodes by migrating some tasks and can process 30 frames generated by depth cameras each second for all three applications. Figure 3 shows the CPU utilization across all PEs with error bars marking standard deviation. Total CPU utilization across all PEs is lower with the ad-hoc implementations due to the CPU saturation at full utilization. QoS manager increases the data processing rate by utilizing idle CPUs in the cluster, hence increasing the utilization average.

VIII. Conclusion and Future Work

This work describes higher level applications and services necessary for a smart home and system level services required to support the higher level functionality. As an alternative to implementation of ad-hoc applications for each functionality, we describe a layered architecture that enforces specialization to increase robustness, modularity, and code reuse. We show the effectiveness of our design by evaluating the QoS Manager module against an ad-hoc implementation. We plan to add priority and timeliness support to our QoS Manager.

References

[1] D. Cook and S. Das, *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2004.

[2] G. Abowd, E. Mynatt, and T. Rodden, "The Human Experience," *Pervasive Computing, IEEE*, vol. 1, pp. 48 –57, jan.-march 2002.

[3] J. Wu, L. Huang, D. Wang, and F. Shen, "R-OSGi-based architecture of distributed smart home system," *Consumer Electronics, IEEE Transactions on*, vol. 54, pp. 1166 –1172, august 2008.

[4] J. Bardin, P. Lalanda, and C. Escoffier, "Towards an Automatic Integration of Heterogeneous Services and Devices," in *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*, pp. 171 –178, dec. 2010.

[5] E. E. Stone and M. Skubic, "Evaluation of an inexpensive depth camera for passive in-home fall risk assessment," in *Pervasive Computing Technologies for Healthcare, 2011 5th International Conference on*, pp. 71–77, IEEE, 2011.

[6] A. Arcelus, M. Jones, R. Goubran, and F. Knoefel, "Integration of Smart Home Technologies in a Health Monitoring System for the Elderly," in *Advanced Information Networking and Applications Workshops*, vol. 2, pp. 820–825, IEEE, 2007.

[7] C. Marie, E. Daniel, and C. Eric, "Elderly Daily Activity Habits or Lifestyle in Their Natural Environments," in *Proceedings of the 4th International Conference on Pervasive Technologies Related to Assistive Environments*, p. 26, ACM, 2011.

[8] H. Zhang, F. Wang, and Y. Ai, "An OSGi and agent based control system architecture for smart home," in *Networking, Sensing and Control*, pp. 13–18, IEEE, 2005.

[9] C. Wu, C. Liao, and L. Fu, "Service-Oriented Smart-Home Architecture Based on OSGi and Mobile-Agent Technology," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 2, pp. 193–205, 2007.

[10] S. Nirjon and J. Stankovic, "Kinsight: Localizing and Tracking Household Objects Using Depth-Camera Sensors," in *Distributed Computing in Sensor Systems (DCOSS), 2012 IEEE 8th International Conference on*, pp. 67 –74, may 2012.

[11] M. Pedrasa, T. Spooner, and I. MacGill, "Coordinated Scheduling of Residential Distributed Energy Resources to Optimize Smart Home Energy Services," *Smart Grid, IEEE Transactions on*, vol. 1, no. 2, pp. 134–143, 2010.

[12] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *Computer Vision and Pattern Recognition*, vol. 1, IEEE, 2005.

Computing the exact worst-case End-to-end delays in a Spacewire network using Timed Automata

Jérôme Ermont, Christian Fraboul
Université de Toulouse - IRIT - INPT/ENSEEIHT
{jerome.ermont, christian.fraboul}@enseeiht.fr

***Abstract*—Spacewire is a real-time communication network for use onboard satellites. It has been designed to transmit both payload and control/command data. To guarantee that communications respect the real-time constraints, designers use tools to compute the worst-case end-to-end delays. Among these tools, recursive flow analysis and Network Calculus approaches have been studied. This paper proposes to use the model-checking approach based on timed automata to compute the exact worst-case end-to-end delays and two case studies are presented.**

I. Introduction

SpaceWire [1] is a communication network for use onboard satellites which has been developed by the European Space Agency and the University of Dundee. It provides high-speed data exchanges, from 2Mbps to 200Mbps, between sensors, memories, processing units and downlink telemetry.

One goal of SpaceWire is to carry both the payload and the command/control traffic instead of using dedicated buses, as MIL-STD-1553 buses, for both of them. These two characteristics need different requirements: low throughput and very strict time constraints for command/control traffic and a sustained high bandwidth for payload. Using point-to-point SpaceWire links, both characteristics are easily satisfied. But SpaceWire is based on a part of the IEEE-1355 standard [2] and is defined to connect several equipments. So SpaceWire is a packet switching network.

Due to the space requirements (an important one is the radiation tolerance), routers have to store a minimal amount of data. To ensure this ability, SpaceWire uses wormhole routing: packets are not stored completely but can be forwarded as soon as the output port is free. If the output port is not free, the packet is blocked. In that case, the packet cannot be transferred from the upstream router blocking other packets. The consequence is a variation of the end-to-end (ETE) delays for the packets. A method to verify that the time constraints are guaranteed must be defined.

A similar problem arises in the context of avionics where an upper bound has to be computed in respect to the certification. For this, different solutions exist. Two of them are based on Network Calculus ([3], [4]) and Trajectories ([5], [6]). However, the obtained upper bounds are pessimistic due to the assumptions made by these two approaches.

Other works have been devoted to compute the exact ETE delays of an AFDX network. Existing model checking approaches ([4], [7]) implement an exhaustive analysis of all the possible scenarios. However, it cannot be applied to AFDX configurations with more than 10 flows (a real one is more than 1000 flows) because of the well-known combinatorial explosion problem. In [8], the authors extend the study by considering the scheduling of the flows in the network. This drastically reduces the number of considered scenarios.

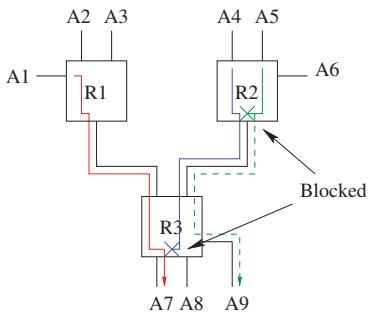


Fig. 1. Wormhole routing

As said before, in SpaceWire, the command/control messages have to be delivered before their deadlines. In [9], the authors propose to compute an upper bound of the worst case ETE delay of each message. Two methods have been studied: one based on Network Calculus and one based on a recursive flow analysis.

Because the size of a SpaceWire architecture is limited, this paper proposes to compute the exact worst case ETE delays using the timed automata theory.

The wormhole routing mechanism is reviewed in Section II. Then, we give the timed automata model of the SpaceWire network in Section III. Two case studies are described in Section IV.

II. Wormhole routing

Memory consumption is an important challenge of space systems because radiation tolerant memories are very expensive. Furthermore, classical routing policies, such as store and forward policy, need to buffer data and cannot be used. In a wormhole routing system, packets are not totally stored in buffers but transmitted, item by item, as soon as the output port is free.

To explain the behavior, let us consider the network architecture in Figure 1. When receiving the first character of a packet sent by the application A1, the router R1 determines the appropriate output port. If the output port is free, the packet is

immediately transmitted to the router R3 and the output port is marked as occupied. The router R3 receives the packet, transmits it to the destination A7 marking the output port as occupied.

Suppose now that the application A4 sends a packet to the application A7 too. The output port of the router R2 is free, the packet is sent to the router R3, marking the output port the router R2 as occupied. The output port of the router R3 is occupied. The packet is then blocked in a small input buffer, 64 bytes in Spacewire, of R3.

And, suppose that the application A5 sends a packet to the application A9. The packet is blocked in the router R2, waiting to the output port occupied by the packet sent by A4.

When the packet from A1 is totally received by A7, the output port of R3 becomes free and the packet from A4 is transmitted to A7. When the output port of R2 becomes free, the packet from A5 will be transmitted.

If two packets are waiting for the same output port, two-level priority queueing is used to reduce the blocking duration of urgent packets. For packets with the same priority level, a simple round-robin procedure is executed to determine which one has to be transmitted as soon as the output port becomes free. This mechanism allows a fair access to the output port but does not guarantee a bounded delay for the transmission of the packets.

In the next section, these characteristics will be modeled in the timed automata theory.

III. Modeling using timed automata

This section proposes a review of the timed automata theory. Then, the modeling of the Spacewire architecture is given. Finally, the method used to compute the worst-case ETE delay is explained.

A. Timed automata

Timed automata have been first proposed by Alur and Dill [10] in order to describe systems behavior with time.
A timed automaton is a finite automaton with a set of clocks, i.e. real and positive variables increasing uniformly with time. Transitions labels can be a guard, i.e. a condition on clock values, actions, updates, which assign new value to clocks.

The composition of timed automata is obtained by a synchronous product. Each action a executed by a first timed automaton corresponds to an action with the same name a executed in parallel by a second timed automaton. In other words, a transition which executes the action a can be fired only if another transition labelled a is possible. The two transitions are performed simultaneously. Thus communication uses the rendez-vous mechanism.

Performing transitions requires no time. Conversely, time elapses in nodes. Each node is labelled by an invariant, that is a boolean condition on clocks. The node occupation is dependent of this invariant: the node is occupied if the invariant is true.

Several extensions of timed automata have been proposed. One of these extensions is timed automata with shared integer

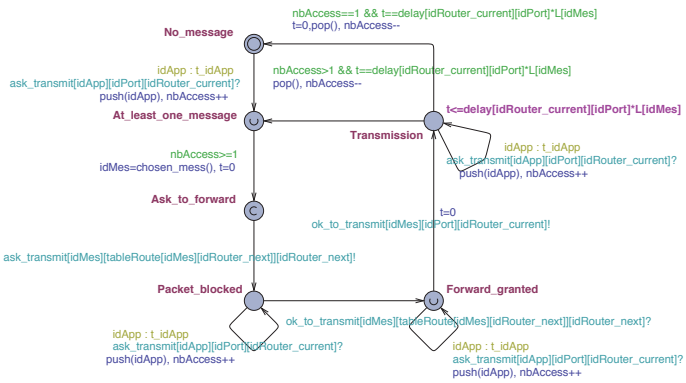


Fig. 2. Timed automaton of an outport of a router

variables. The principle consists in defining a set of integer variables which are shared by different timed automata. Consequently, the values of these variables can be consulted and updated by the different timed automata [11].

A system modeled with timed automata can be verified using a reachability analysis which is performed by model-checking. It consists in encoding each property in terms of the reachability of a given node of one of the automata. So, a property is verified by the reachability of the associated node if and only if this node is reachable from an initial configuration.

The approach that is considered in this paper is based on timed automata with shared integer variables which are represented by nodes of a timed automaton. The modeling of a Spacewire architecture with timed automata is now presented. It is based on Uppaal [11].

B. Modeling a Spacewire architecture

A Spacewire architecture is composed of periodic functions and routers. The timed automata system is then composed of:

- one automaton per periodic function, which generates periodically a packet;
- one automaton per router output port, which models the transmission of packets on the output link, considering the blocking mechanism, the capacity of the link and the length of the message.

Figure 2 is the timed automata model of an output port. When a packet is received by the output port, it is pushed in an input queue corresponding to its priority level. Then, the modeled behavior is as follow:

1) when the output port is free, a packet is chosen considering the policy as explained in Section II. The output port of the router is then blocked for other packets;
2) the system immediately asks to transmit the packet to the next router. This simulates the transmission of the head of the packet to the next router;
3) while the signal *ok_to_transmit* is not received by the automaton, the packet is blocked. In the next router, three cases are possible:
   - the output port is free and the considered packet is chosen, the router sends the signal *ok_to_transmit* and the packet is released;

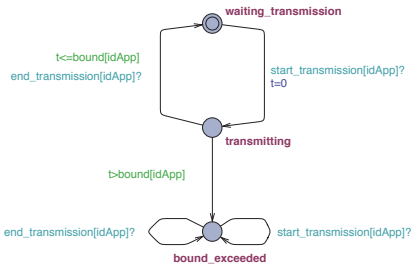


Fig. 3. Test automaton to compute the ETE transmission delay

- the output port becomes free and another packet is chosen, the considered packet is still blocked in all the upstream routers.
- the output port is waiting for the signal *ok_to_transmit* from a downstream router. So, the packet is blocked in the router and all its upstream routers.

This behavior is generalized for all the routers and simulates the progress of the packet item by item in the network;

4) finally, when receiving the signal, the path to the destination is free and the packet is transmitted. The automaton waits for a transmission duration corresponding to the length of the packet (*L[idMes]*) times the capacity of the output link (*delay[idRouter][idPort]*).

The global model is obtained by combining timed automata modeling periodic functions, which are not presented here, and timed automata representing output ports of the routers.

Finally, the worst-case ETE transmission delays can be computed using the model-checking approach.

C. Computing the worst-case ETE transmission delay

The worst-case ETE delay is obtained by model checking. The method consists in verifying that all the packets are received before a bounded delay. The test automata method is used to help the verification process. It consists in verifying if the rejected node of this automaton is reachable or not.

When sending a packet, applications send immediately a signal *start_transmission*, which indicates the beginning of the transmission. The test automaton of the worst-case ETE transmission delay is depicted in Figure 3. The signal *end_transmission* needs to be received before a delay *bound* started when the test automaton receives the signal *start_transmission*. If not, the rejected node *bound_exceeded* is reached and the property is false.

IV. Case studies

In this section, we will present two case-studies.

The first architecture shows the impact of crossed flows and slow links while using recursive flow analysis and Network Calculus (NC) in [9]. We can compare the results obtained by these methods with the exact worst-case ETE transmission delays obtained by model-checking.

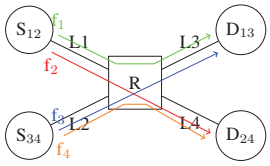


Fig. 4. A first SpaceWire architecture

Flow	Scenario 1		Scenario 2	
L_4 (Mbps)	50		0.2	
	Size	Period	Size	Period
f_1	4000	20	4000	20
f_2	500	8	20	32
f_3	5000	20	5000	20
f_4	400	8	20	32

TABLE I
DIFFERENT SCENARIOS OF THE SECOND CASE STUDY

The second architecture is close to a real industrial Spacewire network which can be used in an observation satellite and shows the limitations of the method.

A. A first architecture: comparison of model-checking with recursive flow analysis and Network Calculus

In Figure 4, the network architecture is composed of 4 applications and a router [9]. The bound of the worst-case ETE delays is computed considering the crossed paths and by varying the capacity of the link L_4. Table I shows the configuration of different studied scenarios and Table II gives the computed ETE delays in ms of the different methods.

In the first scenario, following the remarks in [9], the recursive flow analysis gives the optimal bound but not the NC. And in the second scenario, the capacity of the link L_4 is set to 0.2 Mbps and f_2 and f_4 sends small sized packets. In this situation, NC gives better results than the recursive flow analysis.

For the two scenarios, model-checking gives the exact worst-case ETE transmission delays. They are close to those computed by the recursive analysis in the first scenario. The difference is due to the numeric approximations of the methods. The pessimism of recursive flow analysis and NC can be determined for the second scenario.

The verification of the worst-case ETE delays of each flow f_i takes about 2 minutes on a Macbook with 2.2 GHz Intel Core i7 processor having 8 GB RAM.

B. A second architecture: an industrial-close architecture

A second case study is given in Figure 5. It is composed of 7 sending applications and 3 routers. This is close to an industrial architecture such as the one shown in [9] (9 application units and a processor module which send data and a memory unit and 2 telemetry units which receive data).

In the system of Figure 5, every application A_i sends a packet f_i to the application A_D (destination). The capacity of the links is constant and equal to 50 Mbps. The configuration and the computed worst-case ETE delays in ms are given in table III.

		Scenario 1	Scenario 2
Rec. Analysis	f_1	1.99	10
	f_2	1.99	16.2
	f_3	1.99	10
	f_4	1.99	16.2
NC	f_1	2.08	3.8
	f_2	2.26	4.6
	f_3	2.06	3.8
	f_4	2.06	3.8
MC	f_1	1.98	2.8
	f_2	1.98	3.8
	f_3	1.98	2.8
	f_4	1.98	3.8

TABLE II
RESULTS FOR THE DIFFERENT SCENARIOS

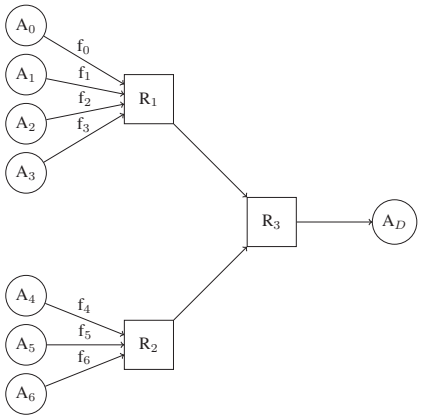


Fig. 5. An industrial-close architecture

The main goal of this architecture is to show the impact of the number of transmitting functions in the system.

The evaluation of the worst-case ETE transmission delays takes about 10 min to be computed on a Macbook with 2.2 GHz Intel Core i7 processor having 8 GB RAM. However, by adding only one more application, the evaluation cannot be performed.

For this architecture, flows f_i arrive synchronously at the output port of routers R_1 and R_2. Model-checking performs an exhaustive analysis of the arrival orders. The number of scheduled transmissions is then too huge to be computed and leads to the well-known combinatorial explosion problem.

But, the timed automata modeling considered here does not take into account the real scheduling of the packets according to the periods of the functions. By doing this, the number of possible scenarios should be reduced and a bigger architecture could be evaluated.

Application	A_0	A_1	A_2	A_3	A_4	A_5	A_6
Period	20	20	20	20	20	20	20
Flow	f_0	f_1	f_2	f_3	f_4	f_5	f_6
Length (bytes)	4000	200	5000	200	4000	5000	4000
WC ETE delay	4.44	4.44	4.48	4.48	3.86	4.48	3.86

TABLE III
CONFIGURATION OF AN INDUSTRIAL-CLOSE ARCHITECTURE AND COMPUTED WORST-CASE ETE DELAYS

V. CONCLUSION

The paper proposes a model-checking approach to compute the exact worst-case ETE delays of Spacewire periodic flows.

Spacewire standard uses wormhole routing to share communications on the network. This mechanism has been modeled using timed automata theory.

The paper proposes then two first case studies to show the feasibility of the computation of the worst-case ETE delays on a Spacewire architecture.

However, even if a Spacewire network architecture is quite small, the well-known combinatorial problem arises: a system with more than 7 sending applications and 3 routers cannot be analyzed due to:

- the scale of time units: few milliseconds for the periods of applications and few nanoseconds for the transmission delays;
- the real number of messages in the network: several packets of a same application can be present in the network due to the period value.

Thus, the number of states considered by the model-checking approach increases with the different possible valuations of clocks and the different possible packet scheduling.

A possible solution to reduce this number could be the one used in [8]. The approach consists in considering only the possible scenarios leading to the worst-case ETE delays. This approach allows to verify AFDX networks with up to 20 flows and should be applied to an industrial Spacewire architecture such as the one presented in [9].

REFERENCES

[1] S. Parkes and P. Armbruster, "SpaceWire: a spacecraft onboard network for real-time communications," in *Real Time Conference, 2005. 14th IEEE-NPSS*, june 2005, pp. 6 –10.

[2] "IEEE Standard for Heterogeneous Interconnect (HIC) (Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction)," *IEEE Std 1355-1995*, p. i, 1996.

[3] J.-Y. Le Boudec and P. Thiran, *Network Calculus*, ser. Lecture Notes in Computer Science (LNCS). Springer Verlag, 2001.

[4] H. Charara, J.-L. Scharbarg, J. Ermont, and C. Fraboul, "Methods for bounding end-to-end delays on an AFDX network," *Real-Time Systems, Euromicro Conference on*, vol. 0, pp. 193–202, 2006.

[5] S. Martin and P. Minet, "Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class," *Parallel and Distributed Processing Symposium, International*, vol. 0, p. 167, 2006.

[6] H. Bauer, J.-L. Scharbarg, and C. Fraboul, "Improving the Worst-Case Delay Analysis of an AFDX Network Using an Optimized Trajectory Approach," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 4, pp. 521 –533, nov. 2010.

[7] M. Lauer, J. Ermont, C. Pagetti, and F. Boniol, "Analyzing End-to-End Functional Delays on an IMA Platform," in *ISoLA*, ser. LNCS, vol. 6415. Springer, 2010, pp. 243–257.

[8] M. Adnan, J. Scharbarg, J. Ermont, and C. Fraboul, "An improved timed automata approach for computing exact worst-case delays of AFDX sporadic flows," in *Emerging Technologies and Factory Automation (ETFA), 2012 IEEE 17th Conference on*. IEEE, September 2012.

[9] T. Ferrandiz, F. Frances, and C. Fraboul, "A Sensitivity Analysis of Two Worst-Case Delay Computation Methods for SpaceWire Networks," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, july 2012.

[10] R. Alur and D. L. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[11] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a Nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1–2, pp. 134–152, 1997.

Consolidate-to-Idle

The second dimension is almost for free.

Marcus Völp, Johannes Steinmetz, Marcus Hähnel
Institute for Systems Architecture, Operating Systems Group
Technische Universität Dresden
Dresden, Germany
{voelp, mhaehnel}@os.inf.tu-dresden.de

***Abstract*—Following time, energy is the second most important resource developers of real-time and embedded systems have to consider. Many solutions have been proposed for exploiting slack along the time axis, e.g., to improve the responsiveness of aperiodic jobs or to reduce the processor frequency and supply voltage. In this work, we focus on the spatial dimension. Our goal is to consolidate a given real-time workload to as few cores as possible to keep the majority of system resources powered off until their unavailability risks missing a deadline. Our proposed scheme is completely independent of the actual multiprocessor scheduling policy and also independent of the actual method for computing slack as long as they reveal the original processor allocation and the slack that is available on each CPU. We started implementing a partitioned EDF based scheduler for Linux using our consolidation scheme. This scheduler will enable us to demonstrate expected energy savings and can be used to evaluate different, consolidation-based scheduling policies.**

***Keywords*—energy; slack time; scheduling; many-core; real-time**

I. Introduction

Energy is a vital resource of real-time and embedded systems, outweighed only by the demand to meet the deadlines of admitted tasks. Many solutions have been proposed to dynamically adjust the speed of processing resources and, as a consequence, the supply voltage while preserving the systems' real-time properties (see e.g., [1]–[3]). However, when we continue to scale to smaller technology nodes, leakage will dominate the energy consumption and the returns from dynamic voltage and frequency scaling will diminish [4]. Clock-gating and the implied disabling of hardware resources will become the dominant mitigation strategy. Some computer architects even predict that 50 % of a fixed-size chip cannot be kept powered up for long periods of time [5] once we have reached 8 nm and below.

This paper presents our early results on a slack-based energy-aware scheduler. Unlike previous approaches, our scheduler does not primarily seek to optimize along the time dimension but instead focuses on the spatial dimension. Our goal is to consolidate a given real-time workload on as few cores as possible, powering up additional resources only if we would otherwise risk missing a deadline. In accordance to race-to-idle [6], we call this approach *consolidate-to-idle*.

To our surprise, consolidate-to-idle needs to know only the current assignment of tasks to CPU cores as it is produced by an arbitrary multiprocessor scheduler, which we call the *underlying scheduler*, and the slack time that is available on each of these cores. In all other respects, consolidate-to-idle is completely independent of this multiprocessor scheduler and of the actual method used for computing this slack. In this respect, once this original scheduler has produced a schedule and computed the slack, consolidate-to-idle comes almost for free. The only two costs that we have to consider are the time required to power up additional CPUs and the overheads that occur when migrating tasks more often than the underlying scheduler.

To enable an in depth evaluation of our approach, we are currently implementing a partitioned EDF (pEDF) scheduler for Linux with consolidation functionality. While the scheduler was not finished in time for this publication we hope to soon be able to present first results of *consolidate-to-idle*.

II. Foundations

To explain consolidate-to-idle, let us start with a rather abstract definition of schedulers. A scheduler decides when and where to run the tasks τ of the task set T. We characterize the behavior of such an arbitrary scheduler by the mapping S, which selects for each point in time t and for each CPU m the task that runs on this CPU. We use the special symbol $\diamond$ to denote that no task is selected and that the CPU is idle at time t. Tasks τ can typically react to previous scheduling decisions (e.g., by yielding until the release of their next job if they did run long enough to complete the current job). We characterize this behavior by a not further specified function ϕ, which given the previous scheduling decisions returns how each task would react (i.e., run, block or yield until the next release), and evaluate ϕ and S in an interleaved fashion. That is, ϕ takes the schedule up to time $t-1$ to produce a reaction at time t and S picks a runnable task or returns $\diamond$ to indicate an idle CPU in the schedule for time t. If S produces such a mapping for task τ at time t (i.e., $S_\phi(t,m) = \tau, \tau \neq \diamond$), we call the CPU m to which S has assigned τ the *home CPU* of this task at time t and write $m = home_{S,\phi}(\tau, t)$.

The amount of time by which the schedule may be deferred before tasks risk missing their deadlines is called the *slack* or slack time σ. Obviously, the slack varies over time as jobs of tasks complete or as we defer the execution of the tasks selected by the underlying scheduler. We write $\sigma(t,m)$ to denote the slack available at time t on CPU m. In other words, if at time t, we have slack $\sigma(t,m)$ on CPU m then we may defer from $S_\phi(t,m)$ until time $t+\sigma(t,m)$ to run other tasks. Notice that $S_\phi(t,m)$ may differ from $S_\phi(t+\sigma(t,m),m)$, for example, because the job of the former task may already be

completed. However, as long as $\sigma(t,m)$ correctly represents the slack, continuing with $S_\phi(t+\sigma(t,m),m)$ will not result in deadline misses.

Our approach is to run τ either during the slack-time of some consolidating CPU or on its home CPU according to $S_\phi(t,m)$. This way, feasibility of consolidate-to-idle follows immediately from the feasibility test for the underlying scheduler respectively from the correctness of the slack computation. In this way, consolidate-to-idle is independent of this underlying scheduling and slack computation method.

For example, a partitioned EDF (pEDF) scheduler assigns each task a fixed CPU m and returns for m the task τ whose current job has the earliest deadline. If we assume periodic tasks τ_i with implicit relative deadlines $D_i := P_i$, characterized as usual by tuples $\tau_i := (P_i, C_i)$ with period P_i and worst-case execution time C_i, $S(t,m)$ will return τ_i if $t \leq r_i + D_i$ (where r_i is the point in time when τ_i's current job was released), if there is no task τ_j with $r_j + D_j < r_i + D_i$, if τ_i is runnable (i.e., $\phi(S^{t-1}, \tau_i, t) = run$), and if τ_i's remaining execution time $e_i \leq C_i$ is positive.

Tia's [7] static slack computation algorithm is one method for determining slack in pEDF. Without loss of generality let jobs indices be sorted by their absolute deadlines (i.e., for two jobs J_j, J_k, $r_j + D_j < r_k + D_k \Rightarrow j < k$). Starting from a precomputed slack table for each CPU, Tia partitions the periodic jobs that are in the current hyperperiod and that have deadlines after t into subsets of jobs with deadlines between $r_{c_i} + D_{c_i}$ and $r_{ci+1} + D_{c_{i+1}}$, where c_i is the current job of task τ_i. The cells $w(j,k)$ of the precomputed slack table denote the minimum of the initial slack $\sigma_l(0,m)$ of all periodic jobs τ_l with deadlines in the range $[r_j + D_j, r_k + D_k]$. That is,

$$\sigma_j(0,m) = r_j + D_j + \sum_{\{k | r_k + D_k \leq r_j + D_j\}} C_k \qquad (1)$$

and

$$w(j,k) = \min_{r_j + D_j \leq r_l + D_l \leq r_k + D_k} \sigma_l(0,m). \qquad (2)$$

The static slack $\sigma(t,m)$ at time t by which the schedule of CPU m may be deferred is given by

$$\sigma(t,m) = \min_{1 \leq i \leq n} w_i(t) \qquad (3)$$

where

$$\begin{aligned} w_i(t) &= w(c_i, c_{i+1} - 1) - I - ST - \sum_{k=i+1}^{n} \xi_{c_k}, \\ w_n(t) &= w(c_n, N) - I - ST. \end{aligned}$$

In this equation, $n = |T_{par,m}|$ is the number of tasks allocated to CPU m, N is the number of the jobs of these tasks in the hyperperiod, I is the total idle time on CPU m relative to the beginning of the current hyperperiod, ST stands for the time stolen since the beginning of this hyperperiod and ξ_k is the completed portion of J_k.

III. Consolidate-to-Idle

The idea behind consolidate-to-idle is rather simple: exploit slack time on a few active cores to run the real-time tasks allocated to other cores, powering up additional cores only to prevent tasks from missing their deadlines. For the remaining discussion, we consider CPUs to be in one of two possible states: *consolidating* and *passive*. Consolidating CPUs run tasks of one or more passive CPUs as long as the slack on these CPUs permits. Once the slack is used up, they run the tasks that the underlying scheduler has assigned to them. Passive CPUs run no task but idle or enter a deep power saving state. In this work-in-progress report, we consider only identical CPUs although we see a huge potential for consolidate-to-idle in a heterogeneous setup. For example, extending the general idea of big-little architectures (as proposed by Kumar et al. [8]) to predictability, we may schedule the real-time workload on the predictable cores and then consolidate them on more efficient but unpredictable cores [1].

A. Passive CPUs

Consolidating CPUs may keep other CPUs passive as long as they finish the tasks of these passive CPUs quicker than their slack time runs out. Let SU_m be the time required to start up CPU m after it has entered a power saving state. Let further $M_m^{passive}(t)$ be the costs for migrating tasks at time t to the passive CPU m. Then clearly, we have to start powering up a passive CPU latest at

$$t_{migrate\_home} := t + \sigma(t,m) - SU_m - M_m^{passive}(t). \qquad (4)$$

Because real-time tasks tend to complete well before their worst-case execution times, it is likely that all tasks will complete on the consolidating CPUs and that start up and migration to passive home CPUs will not be necessary most of the time.

The term $M_m(t)$ in Eq. 4 is to prepare for cache aware worst-case execution time analyses. Typically, migration costs are characterized on a per task basis as direct and indirect costs. Direct costs is the time required to stop and to restart it on its destination CPU. Indirect costs is the time required to transfer state that the task requires but that is not immediately transferred with the task. Most notably these costs are the cache transfers of the task's working set. Consolidation typically affects more than just the migrated task. On the consolidating core, the task may have disturbed the cache working set of local tasks (i.e., tasks whose home CPU is this consolidating core). Let $M_{m_c}^{active}(t)$ capture these costs. On the formerly passive CPU, consolidated tasks had no chance to fetch cachelines required by subsequent tasks. At the same time, they had no chance to interfere with these tasks. Whether $M_m^{passive}(t)$ is positive or negative therefore depends on the consolidated tasks and on the tasks that follow after time t in the schedule of CPU m. Of course, we will have to look at reasonable approximations of $M_{m_c}^{active}$ and $M_m^{passive}$.

As it is our goal to keep passive CPUs off most of the time, computing the slack times and triggering the startup of passive CPUs has to be part of the responsibility of the consolidating CPUs. In this case, we say such a consolidating CPU m_c *observes* the slack of a passive CPU m and collect all CPUs m that m_c observes in the set $O(m_c)$.

B. Consolidating CPUs

In general, the task set T may include real-time tasks whose home CPU is a consolidating CPU m_c. In this case, we must

[1] We assume here that it is possible to limit the interference between these cores in the on-chip network and memory blocks.

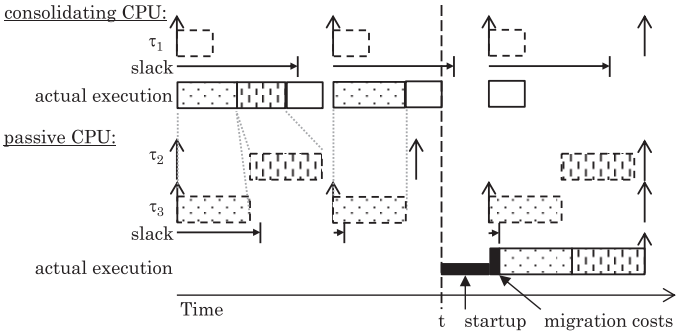


Fig. 1. Tasks of the passive CPU are consolidated to the consolidating CPU. At time t, the slack time on the passive CPU m falls below $SU_m + M_m^{passive}(t)$. We need to power up this CPU.

also consider the slack $\sigma(t, m_c)$ of the consolidating CPU. That is, to not risk missing a deadline, the consolidating CPU must start executing local tasks latest at $t + \sigma(t, m_c) - M_{m_c}^{active}(t)$.

To reduce synchronization and management overhead, we maintain the invariant that a consolidating CPU only runs tasks from passive CPUs in its slack time and only local tasks if no more slack is available. When activating a passive CPU, we already consider the migration overhead of all local tasks that may have been relocated to other consolidating CPUs. Also, we do not have to consider the migration overhead of tasks that we consolidate on such a consolidating CPU because we directly deduce feasibility from the local and remote slack and from the admission for the underlying scheduler. Instead, we execute tasks that we consolidate as fast as we can to cause them to make as much progress as possible.

In general, it would be possible to further defer the activation of passive CPUs if the slack on consolidating CPUs suffices to complete the scheduled passive CPU before its deadline. In our preliminary analysis, we did not consider this possibility. In non-identical settings, exploiting this option requires reevaluating the feasibility of the passive CPU's schedule, an overhead we seek to avoid.

Fig. 1 illustrates our approach on the example of three tasks $\tau_1 \ldots \tau_3$. Shown is the EDF schedule (dashed boxes) and slack (horizontal arrows) as determined by the underlying scheduler. Also shown is the actual execution that happens on the respective CPUs. The CPU slack suffices to consolidate the jobs $\tau_{2,1}$, $\tau_{3,1}$ and $\tau_{3,2}$ together with τ_1 on the consolidating CPU. At time t, we have to start powering up the passive CPU to be ready to execute $\tau_{2,2}$ and $\tau_{3,3}$ on their home CPU. Their execution is prolonged by the migration overhead $M_m^{passive}(t)$. If during τ_1's first period, $\tau_{2,1}$ would have executed longer, $\tau_{1,1}$ would have preempted it at the point in time denoted by the end of the first horizontal arrow when the local slack on the consolidating CPU would be depleted.

C. Algorithm

Before we proceed by sketching the algorithm for consolidate-to-idle, it is helpful for a deeper understanding to cast the above operations into the terminology of Tia's slack computation method (Eq. 3). At consolidating CPUs, we steal the slack of local tasks to run tasks of passive CPUs. Hence, we increase the stolen time ST at the consolidating CPU. On the passive CPUs, we increase the idle time I by keeping them passive. However, at the same time we also complete jobs J_k, whose home CPU is passive, by consolidating them.

```
1  // invoked at the usual scheduling events,
2  // in case σ(t, m_c) = 0 and when starting up
3  // a CPU.
4  schedule:
5    // update statistics for current job:
6    adjust progress of the current job J_k;
7    if (home(J_k) ∈ O(m_c)) then
8      recompute the slack time of home(J_k);
9      setup a timer to
10       σ(t, home(J_k)) − SU_home(J_k) − M^passive_home(J_k)(t);
11   endif
12   recompute the local slack σ(t, m_c);

14   // check whether we need to run local jobs:
15   if (σ(t, m_c) = M^active_m_c(t)) then
16     setup the usual timers (e.g., deadline);
17     switch to S_φ(t, m_c);
18   endif

20   // check whether to deactivate:
21   if (become_passive()) then
22     distribute observed CPUs to the sets
23       O(m_c) of other consolitating CPUs;
24     set state to passive;
25     enter sleep state();
26     goto schedule:
27   endif

29   // keep track of local slack:
30   setup timer to σ(t, m_c);

32   // consolidate task of passive CPU:
33   pick τ = S_φ(t, m′) from some passive CPU m′;
34   switch to τ;

36 // invoked in case σ(t, m) = SU_m + M_m(t):
37 startup_passive:
38   set state of m to active;
39   adjust observation of passive CPUs;
40   startup CPU m;
41   let J_k be the current job on m_c;
42   if (home(J_k) = m) then
43     schedule();
44   endif
```

Fig. 2. Pseudo code of the central functions of consolidate-to-idle. The respective entry points are the call backs of the timers setup to react to slack depletion. As before t is the current time and m_c denotes the CPU executing this code.

Fig. 2 contains pseudo code describing the basic operation of consolidate-to-idle. Lines 5–11 adjust the slack of the consolidating CPU. In addition, if the consolidating CPU has executed a job of a passive CPU, we must also adjust the slack of this CPU. Timeouts are set to the time when the slack of an observed CPU m reaches $SU_m + M_m^{passive}(t)$ because latest at this time, the observing CPU has to start activating m. Lines 14–18 execute local jobs if no more slack remains on the consolidating CPU by switching to the current job selected by the underlying scheduler. The function `become_passive()` determines whether a CPU should become passive or remain

active as a consolidator. In our example, this function always returns false for CPU 0 and true for all others. In general, more sophisticated strategies are imaginable, which consider the number and distance to passive CPUs and the heat that has been building up. In line 33, we pick a task τ to consolidate on m_c if such a task exists. The strategy for selecting tasks from passive CPUs is a further dimension in the design space of consolidate-to-idle that is worth exploring in the future. Sec. III-D1 gives some hints on the implied synchronization overhead. However, a thorough discussion of this point is out of the scope of this work-in-progress report. For our example implementation, we plan to use a Round-Robin like approach, which cycles through passive CPUs picking the current task (if present) and runs it to completion or until some slack is depleted.

D. Practical matters

We would like to raise attention to two practical matters: the interrelationship between the consolidation strategy and run-queue synchronization and the possibility to integrate other slack-based approaches.

1) Run-queue synchronization: Clearly, if multiple consolidating CPUs pick from the same passive CPUs, global synchronization on the run queue of this passive CPU is needed to prevent different consolidating CPUs from picking the same task. Notice that because we deactivate a CPU if it is passive, it will not modify its local run queue. The queue may therefore be accessed by a single consolidating CPU without further precaution or overheads (assuming passive CPUs stay offline most of the time)[2]. At the same time, strategies picking multiple tasks from the same passive CPU are faster in creating more slack on these CPUs by increasing the completed portion of the jobs they execute on their stolen time.

2) Integration of other slack-based approaches: Consolidation decisions are only based on the home CPUs of tasks and on the slack $\sigma(t, m)$ that the used slack-computation method reports to the consolidation algorithm. Reducing this slack by reporting only the time that is left after considering other uses (e.g., executing aperiodic tasks) is an easy way to integrate other slack-based approaches. Ideally, these approaches are aware of the consolidation strategy and report reduced slack only on consolidating (i.e., active) CPUs or they influence the scheduler S in that it reports also the use of this slack, for example, in terms of servers on the corresponding CPUs whose budgets are used to execute aperiodic tasks.

In these integrated scenarios, where slack needs to be determined anyway, the overheads of consolidate-to-idle are limited to the power up times of passive CPUs and to the overheads for migrating tasks from consolidating to recently activated CPUs. In this sense, consolidate-to-idle comes almost for free.

IV. Conclusions and Future Work

This paper has shown early results of consolidate-to-idle, a slack-time based approach to save energy in manycore systems by disabling inactive CPUs, aggregating their real-time tasks to active consolidating cores. We have seen that consolidate-to-idle is completely independent of the underlying scheduler and slack computation method provided they reveal the home CPU of all current jobs and the slack that remains before these jobs must be run. Of course, many directions in the design space of consolidate-to-idle remain open. To avoid contention on the run queue, we have chosen an ad-hoc $n : 1$ association of passive CPUs to consolidating CPUs. Other strategies, such as balancing the slack of passive CPUs by picking possibly several tasks from the one with the smallest slack to consolidate them on different CPUs or picking a task with hot cache working set on the consolidating CPU, might further defer when passive CPUs need to be powered up. Other dimensions include decisions on when consolidating CPUs should become passive and heterogeneity between the resources. Despite these open questions, we confidently believe that there is a case for the spatial exploitation of slack time and that consolidate-to-idle is a first step in this direction.

[2]Notice that local synchronization may still be necessary to avoid races with code that re-enters unblocking tasks to the run queues.

Acknowledgments

This work is in part funded by the DFG through the collaborative research center "Highly Adaptive Energy Efficient Systems" (HAEC) and through the cluster of excellence "Center for Advancing Electronics Dresden" and by the EU and the state Saxony through the ESF young researcher group "IMData". Thanks to Michael Roitzsch for the insightful discussions.

References

[1] J.-J. Che, C.-Y. Yang, and T.-W. Kuo, "Slack reclamation for real-time task scheduling over dynamic voltage scaling multiprocessors," in *Sensor Networks, Ubiquitous, and Trustworthy Computing*, vol. 1, June 2006, p. 8 pp.

[2] R. Jejurikar and R. Gupta, "Dynamic slack reclamation with procrastination scheduling in real-time embedded systems," in *Proceedings of the 42nd annual Design Automation Conference DAC '05*, New York, NY, USA, pp. 111–116.

[3] D. Zhu, R. Melhem, and B. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 7, pp. 686–700, 2003.

[4] E. L. Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *2010 Workshop on Power Aware Computing and Systems (Hot Power'10)*. Vancouver, Canada: Usenix.

[5] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 365–376.

[6] S. Albers and A. Antoniadis, "Race to idle: new algorithms for speed scaling with a sleep state," in *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '12, pp. 1266–1285.

[7] T. S. Tia, "Utilizing slack time for aperiodic and sporadic requests scheduling in real-time systems," Ph.D. thesis (Tech. report No. UIUCDCS-R-95-1906), Dept. of Computer Science, University of Illinois at Urbana-Champaign, April 1995.

[8] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance," in *Proceedings of the 31st annual international symposium on Computer archit ecture ISCA '04*. Washington, DC, USA: IEEE Computer Society.

Highly Efficient and Predictable Group Communication over Multi-core NoCs *

Karthik Yagna, Frank Mueller
North Carolina State University
kyagna@ncsu.edu, mueller@cs.ncsu.edu

ABSTRACT

Massive multi-core embedded processors with network-on-chip (NoC) are becoming common in real-time systems. These architectures benefit real-time scheduling of tasks and provide higher processing capability due to abundance of cores. The core-to-core communication can be leveraged by adopting message passing to further increase system scalability. Despite these advantages, multicores pose predictability challenges.

In this work, we develop efficient and predictable group communication using message passing specifically designed for large core counts in 2D mesh NoC architectures. We have implemented the most commonly used collectives in such a way that they incur low latency and high timing predictability making them suitable for real-time systems. Experimental results on the TilePro64 hardware platform show that our collectives can significantly reduce communication times by up to 95% for single packet messages. In addition, the primitives have significantly lower variance compared to prior work, thereby providing better real-time predictability.

1. INTRODUCTION

The future of computing is rapidly changing as multicore processors are becoming ubiquitous. Massive multi-core platforms with NoC architectures are being employed for real-time systems. These architectures provide a significant advancement due to an abundance of cores. This allows a large number of cooperating tasks to be scheduled together. These tasks can employ group communication via message passing over the NoC to achieve scalability and reduced latency.

However, poor group communication implementations can result in increased and highly variant latency due to NoC contention and results in loss of predictability. Consider the case where tasks on different cores are performing an all-to-all communication using message passing. One way to implement all-to-all is to have one task send its message to all other tasks, followed by the next one and so on. This implementation is not efficient and can be improved by allowing multiple partners to communicate in each round. Yet, such an optimization may lead to contention. For example, consider 9 cores taking part in all-to-all communication as in Figure 1. The task on core 3 is trying to send to the task on core 8, and the task on core 4 is trying to send to the task on core 2. This results in 2 messages, one from $3 \rightarrow 8$ and another from $4 \rightarrow 2$. When sent at the same time, contention on link $4 \rightarrow 5$ results in a delay for one of these messages due to arbitration within the NoC hardware routers. Such situations can be avoided using intelligent scheduling of each round of message exchanges.

Additionally, implementations that do not leverage underlying NoC capabilities result in under utilization of the NoC hardware. Typically, NoC architectures provide multiple message queues and networks. On the TilePro64 [3], there are 4 distinct message queues and 2 distinct networks types available for users. One of them is called User Dynamic Network (UDN), and the other one is called Static Network (SN). UDN uses dynamic routing to forward messages from source core to destination. SN, on the other hand, uses statically configured routes to forward packets received on each link. SN is faster than UDN in terms of packet forwarding speed, but is difficult to program and has route setup overhead. Hence, UDN is used for all core-to-core communication purposes, leaving SN unused. Implementations which can leverage such unused hardware features can intelligently extract additional hardware performance.

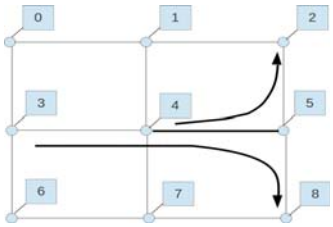


Figure 1: NoC Contention

In our implementation, we employ algorithms to reduce communication overheads and try to use available hardware features to provide better performance. We have currently implemented four commonly used group communications namely Barrier, Broadcast, Reduce and Alltoall [5]. Alltoallv and Allreduce can be built on top of these collectives. Efficient implementation of Alltoall is particularly challenging. Our implementation uses a bottom-up approach in which the communication proceeds from smaller segments to larger segments, but it does not require dividing the grid into smaller submeshes [6]. Other approaches require either dynamic route calculations of offline pre-calculations to store large routing tables [4]. In contrast, our implementation exploits simple pattern-based communication, common in MPI [5] runtime system implementations, to send messages concurrently, yet without contention, to reduce communication latency. This neither requires dynamic computation of a routing schedule nor incurs scheduling overhead or memoization of large routing tables. Our implementation uses message passing over the NoC of a TilePro64, but is generic enough to be adopted to any 2D mesh based NoC architecture.

Experimental results on the TilePro hardware platform show that our implementation has lower latencies and lower timing variability than prior work. We used micro-benchmarks and compared the performance of our implementation against OperaMPI, an MPI implementation for the Tilera platform. Performance improvements of up to 95% are observed in communication for single packet messages with significantly high timing predictability.

2. DESIGN

Our work assumes a generic, generalized 2D mesh NoC switching architecture similar to existing fabricated designs with high core counts [2, 3, 7, 1]. Each core is composed of a compute core,

*This work was supported in part by NSF grants 0905181 and 1239246.

network switch, and local caches. The network switch uses XY dimension-ordered routing to forward messages.

2.1 NoC Architecture

NoC architectures use the network-on-chip to replace the conventional system bus or other topologies of connecting cores. This means that all memory, messaging, and IO communication occur over the NoC, often through physically separate networks to reduce contention. In this work, we focus on building group communication over the messaging network.

2.2 NoC Message Layer

Our implementation provides an MPI type message passing interface on top of NoC. This facilitates basic point-to-point communication used to support our group communication. The NoC message layer implementation optionally provides flow control support. In our design, we turn off flow control when not required by program logic to further improve performance.

2.3 Group Communication Primitives

The key ideas behind our design of group communication primitives are (1) Reduce contention in NoC; (2) Exploit pattern-based communication to exchange messages concurrently; (3) Reduce the number of messages by aggregation; (4) Leverage hardware features to improve performance.

We have used different approaches for each group communication primitive to demonstrate the ways a NoC-based system can support timing reliability and reduced latency.

Alltoall

The Alltoall collective results in all the tasks in the group to exchange messages with each other. In our design, we exploit pattern-based communication to concurrently exchange messages between partners. The entire exchange is split into multiple rounds. In each round, a subset of tasks exchange messages using Manhattan-path (dimension-ordered) routing. The tasks in each round are scheduled in such a way that they do not result in link contention. In each round, the number of hops the message is forwarded to is incremented until all the tasks are covered.

Barrier

A barrier synchronizes a group of tasks. Each task, when reaching the barrier call, blocks until all tasks in the group reach the same barrier call. In order to provide scalable barriers, we designed tree-based barriers that distribute the work evenly among nodes. This also helps in minimizing the cycle differences upon barrier completion. Our design utilizes rooted k-ary trees to this end, where k is configurable. Typically k=3 provides optimal performance.

Broadcast

A broadcast sends a message from the process with rank "root" to all other processes in the group.Tree branches are mapped onto the NoC in a contention-free manner. In our design, we use the SN to implement broadcasts. We designed a tree-based broadcast rooted at the task performing the broadcast. The static route of each task is configured inside the broadcast primitive such that the message from the root flows to each leaf task. To minimize the overhead of route configuration, our design requires only a single route configuration per task, again using contention-free paths.

Reduce

This primitive applies a reduction operation on all tasks in the group and relays the result to one task. We designed our reduce collective similar to the barrier. The reduction operation is performed along the tree. Each task receives values from its children and performs a partial reduction. Tasks then send their partial result toward the root. The root will reduce partial results to obtain the final result.

Alltoallv and AllReduce

Alltoallv is designed as an extension to Alltoall. The AllReduce consists of a reduce followed by a broadcast. Our design follows the same idea.

3. IMPLEMENTATION

This section provides details on the implementation, called NoCMsg, of each group communication primitive. Our implementation of these collectives have an MPI-like API for easy usability.

We implemented the group communication on the Tilera TilePro64. Our implementation is generic and can be extended to any 2D mesh NoC architectures.

3.1 Alltoall and Alltoallv

Alltoall/Alltoallv are the most demanding collectives in terms of network contention, yet they allow flow-control elimination. Based on the particular internal send/receive orders in these collectives, it is possible to guarantee flow-control free communication for transfers between each pair of cores. Further optimization is provided by employing pattern-based communication, which allows several sets of tasks to exchange messages concurrently without contention. The entire exchange is split into multiple rounds.

The rounds are comprised of (1) direct (2) left and (3) right rounds. In direct rounds, each task sends messages only along a straight path to its partner task. In left rounds, each task sends messages along the X direction followed by the Y direction such that their path follows a counter-clockwise direction. In right rounds, each task sends messages along the X direction followed by the Y direction such that their paths follow a clockwise direction. These cases are depicted in Figure 2.

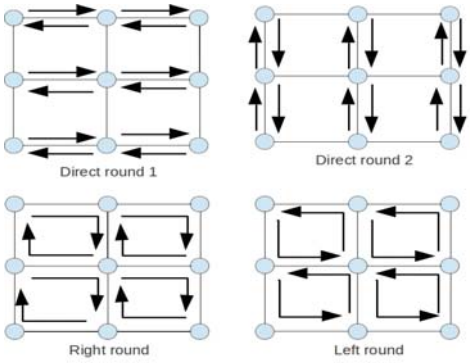


Figure 2: Alltoall Rounds

In each round, the number of hops the message is forwarded is incremented until all the tasks are covered. To begin, each task starts the direct round with one hop. In other words, they exchange messages with their immediate neighbors. Once the round is over, they increment their hop count and exchange messages with a neighbor 2 hops away. This is repeated until the entire width of the grid is covered. After an exchange along the X direction is done, the tasks start direct round 2 and send messages along the Y direction in a similar fashion. This set of rounds is followed by left and right rounds, thereby covering the entire grid. The logic of the algorithm is depicted in the Figure 3.

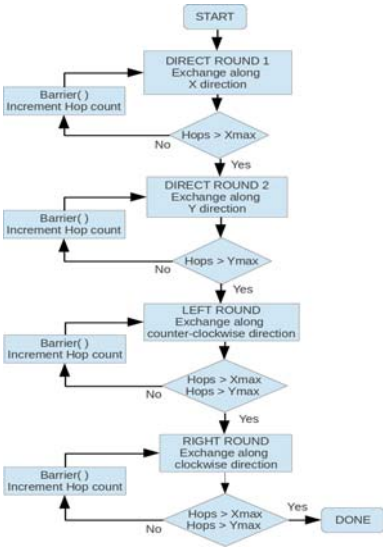


Figure 3: Alltoall Algorithm

3.2 Broadcast

Our Broadcast implementation uses the SN of the TilePro64. The SN is difficult to program and suffers from route setup overhead. However, message forwarding incurs zero overhead (due to a static route setup). Since broadcast has a single sender and multiple receivers, the number of route configurations is low. This was the motivation behind using SN for the broadcast implementation.

We designed a tree-based algorithm rooted at the task performing the broadcast. The route setup in the root is such that the message from the core is sent on its available links. All the tasks in the same column as the root have their route configured such that they receive from the root along the Y direction and send the message along other available links. Tasks in other columns receive along one X direction and send the message along the other X link.

For example, let the task with rank 5 initiate a broadcast. Then, its routes are set up to send the message from the core to all the links. The routes of tasks on cores in column one will be set up such that they send out the received message along the X and Y directions. The routes in all the other tasks will be set up in such a way that they will receive and forward along the X direction. This results in a broadcast tree as shown in Figure 4.

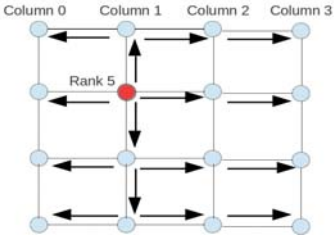


Figure 4: Broadcast Tree: Static Routes Configuration

The static route of each task is configured inside the Broadcast call such that the message from the root flows to each leaf task. Our current implementation requires only a single route configuration per task.

3.3 Barriers

We utilize 3-ary tree-based barriers that distributes the work evenly among nodes to minimize the cycle differences upon barrier completion. The root of this tree is placed in the center of the NoCMsg grid to minimize latency (hops). The process of synchronization involves the children notifying their parents when they have entered the barrier, up to the root. Once the root has received notifications from all children, it broadcasts a notification back down the tree by replying to its children and exits, as do the children. Flow control is not needed in the barrier as the prerequisite of entering into the barrier is that all outstanding sends/receives on the local core have completed. The synchronization packet is small enough to fit into the output queue, *i.e.*, the core can drop an entire synchronization packet into its output queue. It can subsequently begin a blocking send operation that halts the core's pipeline until synchronization packets become available. This technique significantly reduces synchronization costs when all cores are ready.

3.4 Reduce and AllReduce

We designed our Reduce collective similar to the barrier. The reduction operation is performed along the tree. Each child task sends its partial result upward toward the root. The root reduces the partial results to obtain the final result. Our current implementation uses a 3-ary tree rooted at the root of the reduce call.

AllReduce is an extension of Reduce. It is implemented by performing a Reduce relative to the root, followed by a broadcast from the root to all other tasks in the group.

4. EXPERIMENTAL RESULTS

We evaluated our group communication using micro benchmarks on the Tilera TilePro64. We compare the performance of our implementation against OperaMPI, an MPI library specific to the Tilera platform. Each experiment were conducted multiple times to get accurate timing results and variance.

4.1 Microbenchmarks

The micro-benchmarks have a single call to the group communication. In each experiment, we determined the time elapsed in completing the group communication. The basic template of microbenchmark is as shown :

```
NoCMsg_Init(int argc, char **argv)
.........
NoCMsg_Barrier(NoCMsg_Comm comm)
NoCMsg_Timer_start(int timer_num)
NoCMsg_Bcast(void* buffer, int count,
             NoCMsg_Type datatype, int root,
             NoCMsg_Comm comm)
NoCMsg_Timer_stop(int timer_num)
..........
```

The summarized timing results are shown in Table 1 and 2. The plots in Figure 5,7,8 and 6 are timing results for alltoall, barrier, broadcast and reduce micro-benchmark respectively.

Table 1: OperaMPI Execution Times [μsec]

Num tasks	4	9	16	25	36	49
Alltoall	69.57	146.29	250	483.71	759.1	2027.4
Barrier	84.57	174.86	398.57	478.29	679.1	1003
Broadcast	76.29	200.14	337.85	657.43	1026.5	1380.4
Reduce	112.86	232.86	477.71	657.4	955.8	1269.5

Table 2: NoCMsg Execution Times [μsec]

Num tasks	4	9	16	25	36	49
Alltoall	32.28	38.86	114.71	221.29	428.43	761.71
Barrier	3.43	4.43	5.71	10.29	14.17	17.29
Broadcast	3	4	4.43	5.57	7.57	9.14
Reduce	12.57	39.43	27.43	46.83	68.17	87.71

The experimental results follow a similar trend. With increase in number of tasks, the execution time of group communication increases. In case of Opera, the increase in runtime is significant for

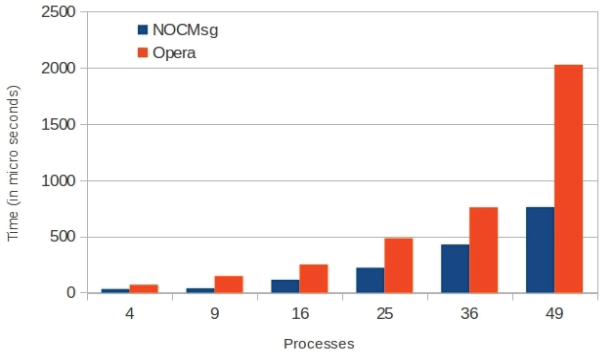


Figure 5: Timing Results for Alltoall

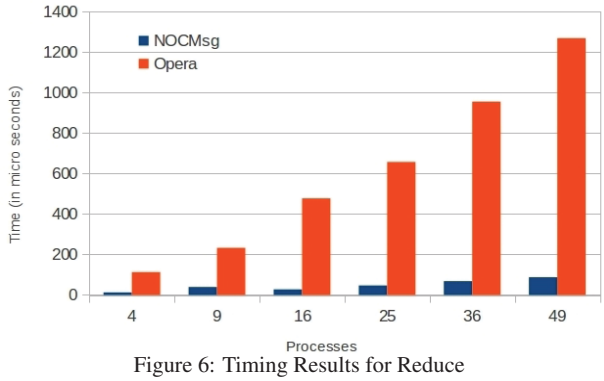


Figure 6: Timing Results for Reduce

larger number of task. In comparison, our implementation is highly efficient and increase in runtime is gradual. Our implementation significantly reduced communication time by up to 95% for single packet messages.

The execution time variance for different micro-benchmark in case of OperaMPI and NocMsg are shown in Table 4 and 3. The variance of timing results for our implementation is several order lower than that of Opera. The difference is significantly large for Alltoall, Broadcast and Barrier case. The lower variance of our implementation results in better real-time predictability making our implementation ideal for real-time applications.

Table 3: NoCMsg Execution Time Variance

Num tasks	4	9	16	25	36	49
Alltoall	0.7	0.4	0.7	5.6	1.3	1.6
Barrier	0.5	0.8	0.4	1.6	1.1	5.6
Broadcast	0	0	0.2	0.24	0.53	0.12
Reduce	11.95	311.39	1.10	183.13	418.13	21.34

Table 4: OperaMPI Execution Time Variance

Num tasks	4	9	16	25	36	49
Alltoall	2.81	983.9	18.2	2276.8	133329.8	622903
Barrier	750.2	302.9	29384.5	1838.2	2910.7	32117
Broadcast	7.3	56.9	259.2	4540.8	3003.7	3869
Reduce	25.2	154.1	19422	4540	12560.9	3725

5. CONCLUSION

We have designed a set of efficient and predictable group communication primitives using message passing utilizing NoC architectures to improve performance and timing predictability specifically design for high-confidence real-time systems. Our implementation of the most commonly used collectives reduced the communication time over a reference MPI implementation by up to 95% for single packet messages. Additionally, the variance of execution times for our implementation is several orders of magnitude

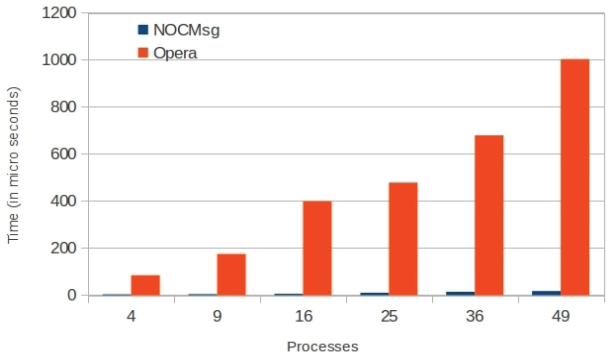


Figure 7: Timing Results for Barrier

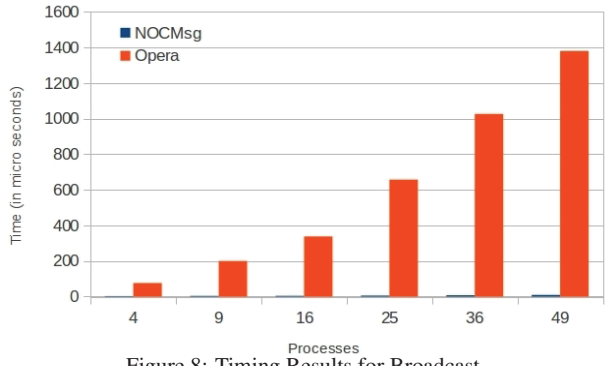


Figure 8: Timing Results for Broadcast

lower than that of the reference MPI implementation, making our implementation ideal for real-time applications.

6. REFERENCES

[1] Single-chip cloud computer. blogs.intel.com/research/2009/12/sccloudcomp.php.

[2] Tera-scale research prototype: Connecting 80 simple sores on a single test chip. ftp://download.intel.com/research/platform/terascale/terascaleresearchprototypebackgrounder.pdf.

[3] Tilera processor family. www.tilera.com/products/-processors.php.

[4] Florian Brandner and Martin Schoeberl. Static routing in symmetric real-time network-on-chips. In *Proceedings of the 20th International Conference on Real-Time and Network Systems*, RTNS '12, pages 61–70, New York, NY, USA, 2012. ACM.

[5] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *European PVM/MPI Users' Group Meeting*, pages 97–104, September 2004.

[6] Young-Joo Suh and Sudhakar Yalamanchili. All-to-all communication with minimum start-up costs in 2d/3d tori and meshes. *IEEE Trans. Parallel Distrib. Syst.*, 9(5):442–458, May 1998.

[7] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27:15–31, 2007.

Knock NOx: Model-based Remote Diagnostics of a Diesel Exhaust Control System

Yash Vardhan Pant Truong X. Nghiem Rahul Mangharam
Dept. Electrical & System Engineering
University of Pennsylvania
{yashpant, nghiem, rahulm}@seas.upenn.edu

***Abstract*—In 2010, over 20.3 million cars were recalled. An increasing percentage of the recalls are due to two reasons: (a) With about 100 million lines of software code and over 60 microprocessors in each car, software bugs in systems such as traction control, stability control, anti-lock brakes and cruise control have accounted for over 13% of the recalls, and are increasing rapidly; (b) With more stringent emissions standards, recalls due to malfunctioning exhaust systems have been escalating and vehicles must be disabled when in violation of emissions guidelines. In this paper, we propose an architecture for interfacing in-vehicle diagnostics with a Remote Diagnostic Center for better fault diagnostics and control code updates, so that expensive, massive recalls due to both software and hardware faults could be avoided in many cases. As a case study, we use the Diesel Aftertreatment system and show how a Remote Diagnostics Center can help in fault isolation. Using a model-based technique and run-time data from the vehicle, we present a method to isolate two common faults on the Diesel Oxidation Catalyst – a part of the aftertreatment system.**

I. Introduction

There has been an increase in recent recalls due to software faults in automotive systems, e.g. Jaguar recalled 17,618 vehicles due to a fault in the cruise control system which did not allow cruise control to be switched off under certain conditions [1]. Recently Ford announced that around 90,000 of its vehicles were susceptible to engine fires because of an under-performing engine cooling controller. Ford's solution to the problem involves recalling the vehicles to the dealership for a software update, which could take several days to manually perform [2].

Typically, technical efforts to avoid these expensive, massive recalls involve significant testing, validation and verification of automotive software and electro-mechanical systems at the design stage. However, current automotive systems lack a systematic approach and infrastructure to support runtime diagnostics of automotive control systems and software in the post-market stage. Once a vehicle leaves the dealership lot, its performance and operation safety are a "black box" to the manufacturers, the original equipment providers and the owner. Thus, as the vehicle's operating state changes due to wear and tear, a problem in a small subset of the vehicles will trigger an expensive full-scale recall of all vehicles of that make/model/year.

Consequently, with the ever increasing functions of automotive control software, there is need to develop an automotive system architecture that supports post-market diagnostics and reprogramming of Electronic Controller Units (ECUs) in the vehicle's control system from a Remote Diagnostics Center (RDC). In this paper, we propose an architecture to tie in-vehicle and remote diagnostics, which involves logging diagnostic data on the vehicle, whenever a fault occurs, and sending it over the cloud to a RDC for comprehensive diagnostics and remote code update. The rest of the paper is organized as follows. We describe the overall design goal of our proposed remote diagnostics and code update approach for automotive systems in Section II. In Section III, we introduce a case study in which a model-based remote diagnostics method is developed for the diesel exhaust control system, which is susceptible to significant recalls. Preliminary simulation results of our method are presented in Section V and we conclude with a discussion on future directions (Section VI).

II. Remote Diagnostics and Code Update Architecture

Monitoring of automotive control systems and software for early detection of faults is necessary. For example, with the stringent emission standards of the future [3], faults need to be detected in the Diesel Aftertreatment (DA) system quickly or the vehicle will violate the emission standards. Standard procedure for dealing with faults in the DA system is to shut the vehicle down and overhaul the DA system, which is both costly and time consuming. While model-based fault detection and isolation techniques could provide more details about the faults to lessen the severity of this standard procedure, they cannot be implemented on the embedded controllers used in production vehicles due to limited processing capabilities of the controllers. Taking into account these factors, we propose an architecture for remote diagnostics and remote code update of automotive control systems, as illustrated in Fig. 1.

On the vehicle side, whenever a fault is detected, the vehicular network (e.g., GM's OnStar [4]) is used to send logged sensor/actuator data and on-board diagnostic (OBD) codes to the RDC. The RDC then uses the received diagnostic data to get more information about the fault, e.g., which component is faulty (fault isolation). Once the fault is isolated, the RDC can either send custom diagnostic codes to the vehicle's control system to get more fault information, or send a code update to reconfigure the controllers to get the system back into the fault-free state (if possible). This allows the vehicle recall process to be less reactive. The logged data can also help identify the conditions that lead to a particular fault occurring across multiple vehicles.

In order to achieve the goals of the proposed architecture, our work focuses on the following tasks:

Fig. 1. End-to-end stages of the proposed automotive architecture. (1) When an unexpected fault is reported, the remote datacenter (RDC) sends custom diagnostic code to the vehicle to observe its performance. Using the vehicle models developed at design-time, the RDC is able to extract the new model for the vehicle (perhaps with changes due to wear and tear, faulty sensors, changes in suspension). (2-3) Using the updated vehicle model, the control system design is reformulated to the faults in the vehicle. (4-5) The RDC remotely updates and verifies the correctness and safety of the reformulated control software.

1) Develop an in-vehicle and RDC networking architecture to allow data transmission between the RDC and multiple vehicles.
2) Modify ECU software architecture to allow for in-software hooks to run custom diagnostic or fault tolerant tasks.
3) Develop model-based diagnostics to use the logged data for fault detection/isolation at the RDC.
4) Reprogram the ECUs over the air with code updates from the RDC.
5) Verify at runtime the updated software and its safety critical functioning.

In this paper, we present our effort for task 3. Specifically, as a case study, we will develop a model-based scheme to isolate faults in the Diesel Oxidation Catalyst (DOC), a key component of the DA system, at the RDC using logged data from the vehicle. The aim of our work is to show how the remote diagnostics architecture can improve the current state of vehicle recalls and repair.

III. Diesel Aftertreatment System

The diesel aftertreatment system is a key component of the vehicle's emission control system and is responsible for reducing the post-combustion engine-out emissions from the vehicle to the atmosphere. The DA system is tasked in particular with reducing the amount of Nitrogen Oxides (NOx), Carbon Monoxide (C), Hydrocarbons (HC) and particulate matter (PM) being released to the atmosphere. Fig. 2 shows a typical DA system architecture. In diesel vehicles, the DA system has a multi-step approach to first chemically oxidize the hydrocarbons into water and carbon dioxide (at the DOC), then reduce the oxides of Nitrogen (NOx) to ammonia and carbon dioxide (at the SCR) and finally burn the particulate matter (at the DPF).

In this work, we will focus on fault isolation in the hardware side of the DOC. The diesel oxidation catalyst is responsible for the following tasks

- Oxidizing engine-out carbon-monoxide (CO) to carbon-di-oxide (CO_2).

$$CO + \tfrac{1}{2}O_2 \rightarrow CO_2 \tag{1}$$

- Oxidizing the unburnt fuel (hydrocarbons) to carbon-di-oxide and water (H_2O).

$$C_3H_6 + \tfrac{9}{2}O_2 \rightarrow 3CO_2 + 3H_2O \tag{2}$$

- Oxidizing the engine-out Nitrogen monoxide (NO) to Nitrogen-di-Oxide (NO_2).

$$NO + \tfrac{1}{2}O_2 \rightarrow NO_2 \tag{3}$$

A. Common faults in the DOC

The most common faults encountered in a DOC are:

1) **Catalyst Poisoning:** Due to sulphur and other impurities in fuel, compounds formed by them during combustion in the engine pass on to the DOC and take up active sites on the catalyst surface and reduce the ability of the DOC to perform the oxidation reactions. Catalyst poisoning due to Sulphur is reversible to some extent by increasing the DOC temperature.
2) **Sintering:** Sometimes DOC temperatures can go as high as 800 degrees Celsius. At such high temperatures the catalyst *sinters*, or accumulates together, reducing the active surface area, and hence the DOC performance. Unlike Sulphur poisoning, sintering is irreversible and the DOC needs to be replaced if it is sintered.

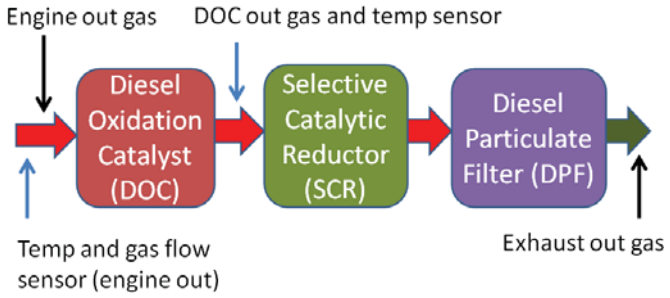


Fig. 2. An architecture for a diesel aftertreatment system. The DOC inlet is where the engine exhaust gases go through to the DOC.

3) **Leak in the DOC inlet:** A leak in the inlet which connects the engine exhaust to the DA system (and the DOC) can occur due to mechanical wear and tear of the system, which means that untreated engine exhaust gases are released into the atmosphere, or worse, into the vehicle's cabin.

In production vehicles, detection of DOC faults is based on the temperature dynamics of the DOC. Temperature sensors upstream and downstream the DOC provide an indirect measure of the rate of reactions occurring inside the DOC. This is because the oxidation of hydrocarbons (HC) is an exothermic reaction, and the more engine exhaust HC is pumped into the DOC, the higher the DOC temperature goes. The DOC temperature is also a function of the engine exhaust gas temperature. Based on this, a look up table for DOC temperature vs engine exhaust temperature and gas flow rate is used to detect faults in the DOC. If for some engine operating condition, the actual DOC temperature is less than the expected DOC temperature, a fault is detected. In case of the DOC, this is the generic OBD-II code corresponding to "DOC efficiency below threshold". Note, this can occur in case of all three faults discussed above, and for actual fault isolation the DOC needs to be tested at the service center. In some cases a non-faulty DOC can be discarded and replaced, yet the fault persists.

The next section will introduce a model based technique which can isolate between these DOC faults using logged data from the vehicle. In particular, we will deal with the case of a leak in the DOC inlet, occurring as a step fault at 500s and resulting in half the engine out exhaust gas not reaching the DOC. The temperature profile for such a fault for one particular operating condition is shown in Fig. 3.

B. Model based design of the DOC

For model based diagnostics of the DOC, we need to model the DOC from first principles. To obtain the model, we assume that all the reactions follow an Eley-Rideal [5] mechanism (where O_2 is adsorbed on the catalyst (Platinum) surface). While the system has been modeled in detail, the main concepts involved were a) First-order rate reactions and Arrhenius-dependence of rates on temperature b) Modeling effects of the inputs to the DA system via a continuously stirred tank reactor (CSTR) approach. A similar modeling approach can be found in more detail in [5]. In general, the final model obtained can be represented in non-linear state space form as

$$\dot{x} = Ax + Bu + f(x) \tag{4}$$

where state-vector $x = [C_{O_2} \quad \theta \quad C_{co} \quad C_{hc} \quad C_{no}]'$ and input vector $u = [C^{in}_{O_2} \quad C^{in}_{co} \quad C^{in}_{hc} \quad C^{in}_{no}]'$, the output of the system are all the states except θ, the catalyst surface coverage fraction. Here, C_i is the concentration of i^{th} gas species and C^{in}_i is the inlet concentration of gas species i to the DOC for $i = \{O_2, CO, HC, NO\}$. A,B are time-invariant matrices and $f(x)$ is the nonlinear part of the system.

IV. Observer based diagnostics for DOC fault isolation

The DOC faults can be broadly classified into two categories, system faults (poisoning and sintering) and actuator faults (leak in DOC inlet). In our model, system faults are characterized by a change in Θ, which is the capacity of the catalyst (Pt) to hold O_2. As active surface area reduces due to both poisoning and sintering, system faults can be detected by identifying the change in Θ from its nominal no-fault value (assumed to be known apriori). On the other hand, a leak in the DOC inlet corresponds to a loss in the gas flow reaching the DOC. Assuming all gas species are lost by an equal factor, let δ denote the fraction of gas lost in the leak. The amount of gas reaching the DOC is $(1-\delta)u$, while the engine exhaust out measurement of gas species is still u (as the loss is not measurable by sensors directly), where u is the input vector as defined in the previous section.

Isolation of the fault to either actuator or plant has been well studied in literature [6], but under the assumption that only the fault we are looking for can occur. The method we propose can isolate either fault and depends on data logged from the vehicle's DA system (sensors and actuators), which we will process at the remote diagnostics center. From Thau's theorem for non-linear observers [7], we can construct a state-estimator of the form

$$\dot{\hat{x}} = (A - LC)\hat{x} + (1-\delta)Bu + f(\hat{x}) + Ly \tag{5}$$

where C is the observation matrix, $\hat{x}$ is the state-estimate and $\hat{y} = C\hat{x}$ is the output estimate for the observer. Note, that the A matrix is a function of Θ and under no fault conditions, $\delta = 0$.

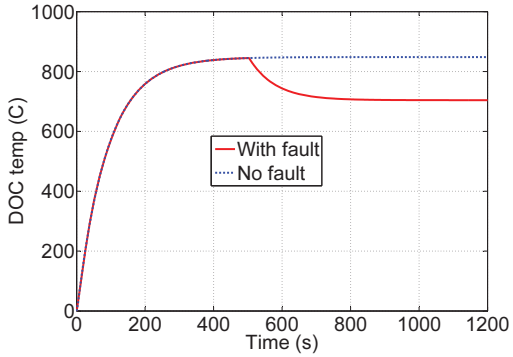


Fig. 3. Temperature measurement at DOC outlet for a DOC inlet leak fault (step-fault, at 500s) and the no-fault temperature measurement at DOC outlet. The inputs to the DOC were the HC gas inlet concentration (subject to fault), and the engine exhaust-out gas temperature was constant at 450 Celsius.

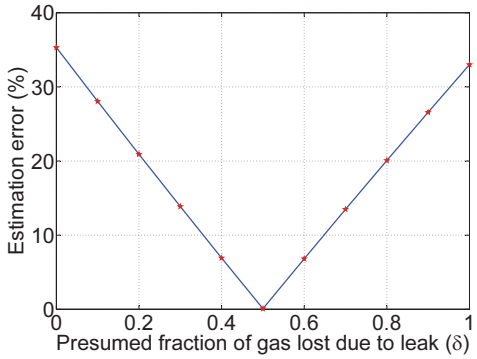


Fig. 4. The difference in observer estimates and measured DOC outputs grows as δ is varied. Note, the actual fault corresponded to $\delta = 0.5$, and the estimation goes to less than 1% when we run the observer for $\delta = 0.5$, indicating that the fault is indeed a leak in the DOC inlet.

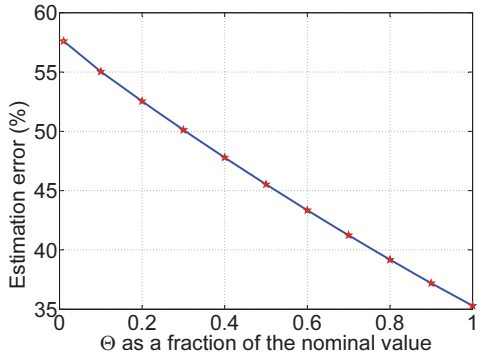


Fig. 5. The difference in observer estimates and measured DOC outputs changes as Θ is varied.

Fault detection for DOC is based on the temperature measurements, and is already a well studied topic and implemented in production vehicles. Assuming that the fault detection is in-vehicle, we use this observer (Eq. 5) for fault isolation at the RDC. The steps for fault isolation are as follows

1) Once a fault occurs, data from the vehicle (both before and after the fault occurs) is sent to the RDC.
2) At the RDC, we simulate the observer in Eq. 5 using logged measurements for two test cases:
   a) For no fault value of Θ and for different values of $\delta \in [0, 1]$.
   b) For $\delta = 0$ and for different values of $\Theta \in [0, \Theta_0]$, where Θ_0 is the no fault value.
3) Whichever case leads to the best match of logged data to observer's estimate is further simulated at a finer granularity (of Θ or δ) until the estimation error is within the desired tolerance.
4) If the first case leads to the better estimation, the fault is an actuator fault. If the second case leads to a better estimation, the fault is a system fault.

V. Preliminary results

We simulate the DOC for a leak in the DOC inlet (corresponding to $\delta = 0.5$). Figure 3 shows the effect of the fault on the temperature of the DOC, and the fault is detected in-vehicle. The logged data for the entire 1200s simulation shows that the fault occurred at 500s. We first formulate the observer for the no fault case. Before the fault, the observer estimates differed from the measurements by $2.28 * 10^{-4}\%$, while for after the fault, the difference was 35%. Following the steps outlined in the previous section, figure 4 shows the results from the first test. Figure 5 shows the results for the second test.

From these results, it is clear that the simulations where we vary δ explain the post-fault system behavior than when we vary Θ. This suggests that the fault is an actuator fault and not a system fault. It is also seen from figure 4 that the least estimation error is for $\delta = 0.5$, which is what the fault was simulated for in the first place. In the case of an actuator fault like a leak, there is no other option but to recall the vehicle for repair, but the correct isolation means that the entire DA system does not need to be tested or the DOC replaced.

VI. Conclusions and future work

In this paper, we outlined an architecture for remote diagnostics of automotive systems and use a test study on the diesel aftertreatment system to show the benefits of this methodology. As shown in Section V, we can correctly the isolate the cause of the fault using logged data from the vehicle. In case the fault was a system fault like poisoning of the catalyst, we could send a code update to the DOC controller to pump in more HC to raise the DOC temperature to reverse the poisoning effects.

For the proposed remote diagnostics approach to be successful, it will be necessary to address the safety and security issues within the architecture. We aim to study these issues as the overall architecture develops further. Future efforts will also focus on dealing with more types and faults and development of optimized algorithms for fault isolation so that the diagnosis of multiple vehicles can be done at the RDC within a short period of time. We also plan to focus on the scheduling of fault diagnosis/tolerance tasks sent from the RDC to the vehicle ECUs.

References

[1] "Jaguar software issue may cause cruise control to stay on." [Online]. Available: http://spectrum.ieee.org/riskfactor/green-tech/advanced-cars/jaguar-software-issue-may-cause-cruise-control-to-stay-on

[2] "Ford announces software fix for 2013 fusion, 2013 escape fire risk," http://wot.motortrend.com/ford-announces-software-fix-for-2013-fusion-2013-escape-fire-risk-302639.html.

[3] "Exhaust emission standards." [Online]. Available: http://epa.gov/otaq/standards/heavy-duty/hdci-exhaust.htm

[4] H. Books, *Articles on Vehicle Telematics*, 2011. [Online]. Available: http://books.google.com/books?id=RlL-ygAACAAJ

[5] D. Upadhyay and M.V. Nieuwstadt, "Modeling a Urea SCR catalyst with Automotive Applications," *Proceedings of the ASME International Mechanical Engineering Congress and Exposition*, 2002.

[6] R. Patton and P. Frank and R. Clark, *Fault Diagnosis in Dynamic Systems*. Prentice Hall, 1989.

[7] F. Thau, "Observing the state of non-linear dynamic systems," *International Journal of Control*, vol. 17, pp. 471–479, 1973.

Model-driven Performance Analysis and Deployment Planning for Real-time Stream Processing

Kyoungho An and Aniruddha Gokhale
ISIS, Dept. of EECS, Vanderbilt University, Nashville, TN 37235, USA
Email: {kyoungho.an, a.gokhale}@vanderbilt.edu

***Abstract*—Real-time stream processing in the cloud is gaining significant attention for its ability to mine massive amounts of data for a variety of applications, such as in reconnaissance missions or search-and-rescue operations. In cloud-based real-time streaming applications, dynamic resource management mechanisms are needed to support the real-time requirements of these applications. However, for any dynamic resource management technique to work, there is first a need to understand the controllable properties (or parameters) of the stream processing applications. Pinpointing these properties and separating them from the application-specific properties that cannot be controlled is hard and requires a scientific approach to obtain these insights. This paper presents a model-driven performance analysis approach for real-time streaming applications to pinpoint their controllable properties. The same modeling framework then makes deployment planning decisions, which is one dimension of dynamic resource management. The presented research is part of our larger effort towards a holistic framework to support real-time and dependable cloud-based applications.**

***Index Terms*—model-based performance analysis and deployment, real-time data processing, cloud computing.**

I. Introduction

Recent trends indicate an increased demand for real-time stream processing in the cloud involving massive amounts of continuous streams of data. For example, in military-based reconnaissance missions or in search-and-rescue operations, there is a need for real-time processing of massive amounts of continuous, incoming streams of data to identify specific enemy targets or survivors, respectively. A number of stream processing platforms have been developed for distributed, real-time stream processing, such as Storm [1], S4 [2], and Flume [3]. The design of these platforms is inspired by Hadoop to process large-scale data sets, however, they are developed to accomplish real-time stream processing unlike batch processing in Hadoop.

Meeting the real-time requirements of the stream processing tasks requires an assurance of predictable, end-to-end execution times from the infrastructure that hosts the different tasks comprising the distributed stream processing activity, which in turn requires effective dynamic resource management. Despite the availability of sophisticated stream processing frameworks, such as Storm, which aim to process data streams in real-time by parallelizing the processing, assuring such bounds through effective dynamic resource management is hard for a variety of reasons. First, the task execution times depend on the input data size, capacity of physical machines that host these tasks, and the number of threads used in concurrent processing. Additionally, queuing delays in the network that hosts the distributed, communicating tasks are not predictable causing further difficulty in bounding the end-to-end execution times. Finally, a lack of effective mapping (*i.e.*, deployment or placement) for allocating the stream processing framework components to the compute resources (*e.g.*, virtual machines in the cloud) gives rise to additional unpredictable performance bottlenecks in processing the streams. All these factors degrade latencies and throughput of the systems in unpredictable ways, and makes it hard to design effective dynamic resource management mechanisms.

More often than ever these problems are handled by a trial-and-error approach. However, such an approach is inefficient, non-scientific, not reusable, and does not provide a dependable way to meeting the end-to-end real-time properties of the applications. For effective dynamic resource management, it is important to understand in what way can the application be dynamically controlled so that the real-time properties of the application can be met. This elicits the need to pinpoint and separate the dynamically controllable properties of these applications from the non-controllable properties. Non controllable properties are those that are imposed by the application logic, which in turn dictates a specific structure to the way stream processing blocks are composed. However, many other factors, such as deployment decisions allocating stream processing blocks to physical hosts and in turn the virtual machines, tuning the infrastructure that hosts these processing blocks, and configuring the network paths are all controllable properties.

An ad hoc approach to pinpointing the controllable properties is not a dependable solution. We surmise that a scientific approach based on model-driven performance analysis and deployment planning (shown in Figure 1) for distributed real-time stream processing may provide the desired solution to better understand these performance-related issues so that subsequently effective dynamic resource management mechanisms can be designed based on the insights gained. Model-based performance analysis has hitherto been applied in many domains. In our prior work [4], we applied model-driven engineering (MDE) [5] for performance analysis of reconfigurable conveyor systems in the context of variability in the physical layouts. Moreno, et al. [6] describe a general approach for

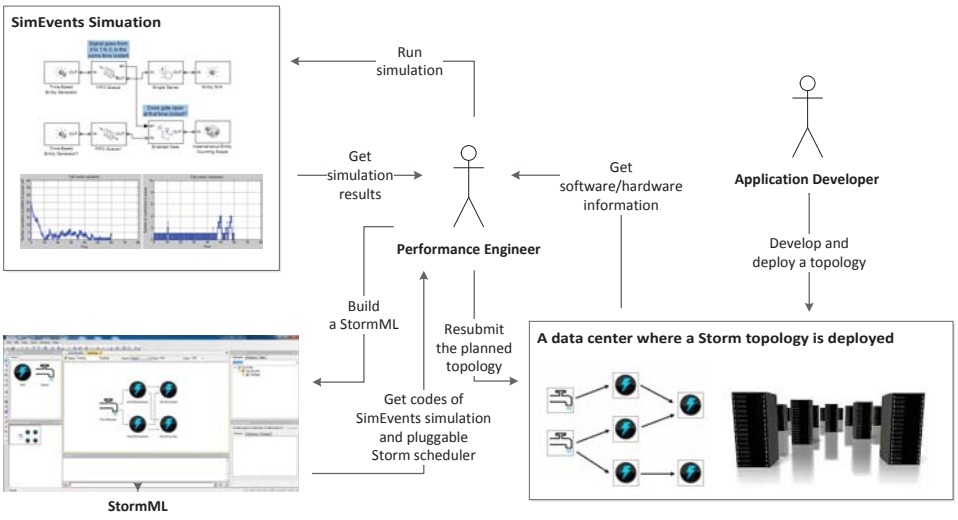


Fig. 1. Model-based Process of Performance Analysis and Deployment Planning

exploiting MDE for software performance analysis. MDE is not only applied in software analysis, but also deployment and configuration. In our prior work [7], [8], we showed how model-driven engineering is used to deploy and configure distributed real-time and embedded (DRE) applications.

For this paper we have focused on the Storm [1] stream processing framework. Our approach is based on profiling several different Storm-based stream applications with different topologies, where a topology in Storm parlance is the structural composition of Storm processing blocks each of which performs some stream processing task. In this context, our MDE-based solution comprises the following artifacts: first, a domain-specific modeling language (DSML) [9] defined for Storm provides intuitive abstractions to performance engineers to run simulations and to deploy their proposed topologies. Second, using the models defined in the DSML, performance details such as bottleneck points, throughput, and latency of each software component, and end-to-end latency of streams are analyzed by automating the execution of a discrete event simulation and obtaining the insights. Third, generative capabilities of the DSML are used to automate the overall analysis and the deployment process.

The rest of the paper is organized as follows: Section II describes our model-based performance analysis approach; and Section III offers concluding remarks alluding to future work.

II. Model-based Analysis for Storm

This section describes our model-based process for analyzing the performance bottlenecks in real-time stream processing applications implemented in Storm.

A. Background of Storm

For our research we have used the Storm stream processing framework. Applications in Storm use two specific building blocks or Executors called Spouts – which are the source of data streams, and Bolts – which process the data streams, may perform operations such as filtering and join, and may produce other streams. Spouts and Bolts can be composed in various configurations to form Storm application topologies. The connections between Spouts and Bolts can be defined based on grouping strategies. A topology can be arbitrarily complex. These logical abstractions must be deployed on hardware resources called a Storm cluster, which is made up of two kinds of hardware nodes: Nimbus (master node) and Supervisor (worker node). Nimbus is responsible for distributing code as well as assigning tasks to Supervisors. Supervisors execute software components of a topology.

Because performance of a Topology can be affected by hardware specifications, each Supervisor contains its hardware specifications as attributes such as the number of CPUs, memory size, and network bandwidth. Each Supervisor includes Slots where worker processes execute. Each Slot is differentiated by its port number. A worker process executes a subset of a specific topology and runs one or more executors.

B. StormML: MDE Framework for Storm

Figure 1 shows the overall process of performance bottleneck analysis and deployment planning for Storm applications using our MDE framework called StormML. First, an application developer develops a Storm-based application, which is then deployed in a Storm cluster by the Storm's default scheduler. The Storm's default scheduler uniformly uses resources by ordering supervisors in terms of available slots. Note that this default deployment may not necessarily provide the best performance. Next, to conduct performance analysis, a Storm model is built using our DSML using data from performance profiling of the test execution. Once the Storm model is built, a Simulink SimEvents model from the

original model is generated by the GME interpreter to run discrete event simulations. Once the SimEvents simulation completes, performance engineers can identify the software components that are bottlenecks, and overall application's throughput and end-to-end latency. Based on the simulation result, the performance engineers can suggest a new deployment plan to improve performance and run another simulation. Through an iterative process, if an optimal deployment plan is determined, a Storm topology is resubmitted to a cluster to run.

The Generic Modeling Environment (GME) [10] is used to develop the DSML named StormML and generative capabilities for StormML. Figure 2 illustrates the StormML meta-model, which is at the heart of the DSML. The StormML meta-model consists of the first class concepts of Storm cluster for hardware components and Storm Topology for software components. For analyzing a topology, we have defined profiler performance metrics such as the number of tasks, average execution time, average input size, average input rate are defined as attributes in the Executor model. In the Connection models, a grouping is defined as an attribute because Storm provides several grouping mechanisms for routing output streams differently.

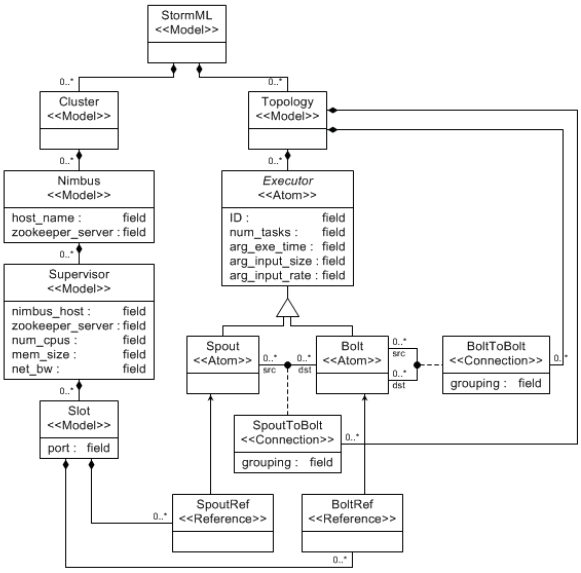


Fig. 2. Meta-model of StormML

To find an optimal deployment, the StormML provides modeling components describing how software is mapped to hardware. We used the GME Reference feature to refer entities defined in one model to be referred in another. Spout and Bolt-based software topologies have reference components, and the reference components are contained in Slots under Supervisors as it is referred as working slots. Our StormML runs a Storm topology which is deployed by Storm's default scheduler and retrieves information defined in the meta-model.

Figure 3 shows a model of a topology defined using StormML for a canonical example of word counting that executes in a Storm cluster. In the example, input streams flow into the topology via a Spout named WordReader. Next, output streams of the Spout flow in two Bolts named WordNormalizers, where sentence streams are split into words. A Shuffle grouping (*i.e.* which is a routing strategy) is used for the stream to balance the stream into multiple Bolts. After the word normalizing process, output streams of each WordNormalizer are sent to the next Bolts named WordCounters, which finally count the words.

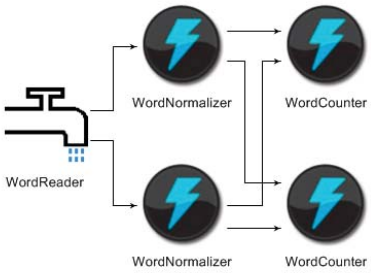


Fig. 3. StormML for Word Count

Figure 4 illustrates the generated SimEvents model for the Word Count example. In the generated model, the WordReader generates a stream periodically by a defined interval. OutputSwitch distributes a stream into two input queues of WordNormalizers in a round robin fashion. Each Bolt has its own FIFO queue. WordNormalizers contain OutputSwitch because processed tuples should be sent to multiple Bolts. In our model, OutputSwitch also distributes tuples in a round robin fashion. Such a distribution may be changed in the SimEvents model depending on what is the grouping strategy used (*e.g.*, Shuffle or Field are one of many groupings provided by Storm that defined how streams are routed in a topology). The total throughput and end-to-end latency can be computed using timer components: StartTimer and ReadTimer. Moreover, to find bottleneck Bolts, server and queue components offer statistical data.

III. Concluding Remarks

The paper described a model-driven tool for analyzing and deploying real-time stream processing applications so that performance bottlenecks can be pinpointed, and subsequently dynamic resource management solutions can be defined. In the current state, the overall process and meta-model of the tool has been developed. Our ongoing work is implementing GME interpreters to generate SimEvents models from StormML and implementations of Storm's pluggable schedulers from StormML. Currently, StormML and SimEvents models are manually created. If interpreters are completed, hardware and software models for StormML are automatically generated by the interpreters from results of test operations. Moreover, the interpreters will transform the StormML into a SimEvents

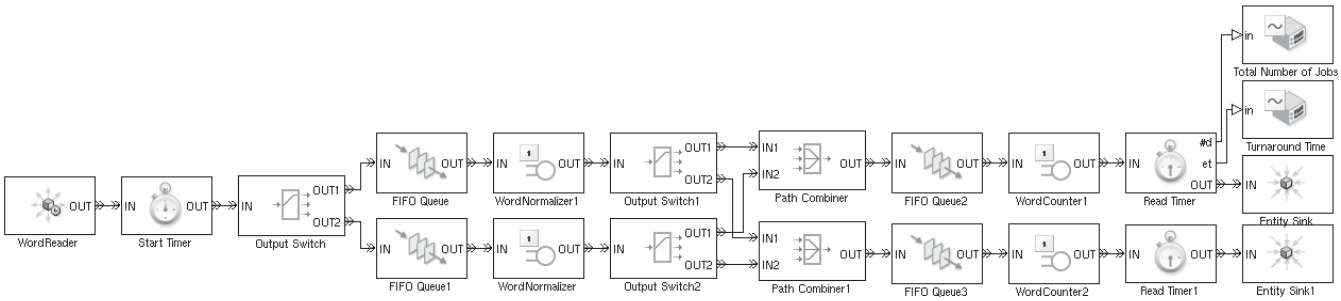


Fig. 4. SimEvents Model for Word Count

model to run discrete event simulation. As a result, through performance evaluation by simulations, an optimal deployment plan is decided by performance engineers and a Java implementation of a pluggable scheduler is made based on the determined deployment plan.

Once the GME interpreters are implemented, we need to refine generated SimEvents models to make it more realistic to actual running Storm applications. In Storm, there are various groupings in Storm such as All grouping, Global grouping, Fields grouping, and Shuffle grouping. In the word count example, only Fields grouping and Shuffle grouping are used, and our current model does not consider that the number of tuples sent to multiple executors is different because of different word frequency. Statistical data of input sizes and input arrival rate for each Bolts should be collected from test operations and applied as parameters in simulation models to refine simulation results.

Moreover, we would like to improve our modeling application to automatically find out an optimal deployment plan for users instead of manual analysis. There have been various research conducted in deployment optimization problem with diverse techniques like genetic algorithms and constraint satisfaction problems (CSP) [11], [12]. In our prior work, we applied a hybrid algorithm that combines worst-fit bin packing with evolutionary algorithms (genetic and particle swarm optimization) for maximizing service uptime of smartphone-based DRE systems [13]. In the work, we extended a framework for spatial deployment algorithm called ScatterD [14]. Likewise, we can extend and apply existing solving techniques and frameworks to find an optimal hardware/software mapping for real-time stream processing.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation NSF SHF/CNS Award CNS 0915976 and NSF CAREER CNS 0845789. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] "Storm," https://github.com/nathanmarz/storm/wiki.
[2] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*. IEEE, 2010, pp. 170–177.
[3] "Apache Flume," http://flume.apache.org.
[4] K. An, A. Trewyn, A. Gokhale, and S. Sastry, "Model-driven Performance Analysis of Reconfigurable Conveyor Systems used in Material Handling Applications," in *Second IEEE/ACM International Conference on Cyber Physical Systems (ICCPS 2011)*. Chicago, IL, USA: IEEE, Apr. 2011, pp. 141–150.
[5] D. C. Schmidt, "Model-Driven Engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, 2006.
[6] G. Moreno and P. Merson, "Model-driven performance analysis," *Quality of Software Architectures. Models and Architectures*, pp. 135–151, 2008.
[7] A. Gokhale, B. Natarajan, D. C. Schmidt, A. Nechypurenko, J. Gray, N. Wang, S. Neema, T. Bapty, and J. Parsons, "CoSMIC: An MDA Generative Tool for Distributed Real-time and Embedded Component Middleware and Applications," in *Proceedings of the OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture*. Seattle, WA: ACM, Nov. 2002.
[8] T. Lu, E. Turkay, A. Gokhale, and D. C. Schmidt, "CoSMIC: An MDA Tool suite for Application Deployment and Configuration," in *Proceedings of the OOPSLA 2003 Workshop on Generative Techniques in the Context of Model Driven Architecture*. Anaheim, CA: ACM, Oct. 2003.
[9] M. Mernik, J. Heering, and A. M. Sloane, "When and How to Develop Domain-specific Languages," *ACM Computing Surveys*, vol. 37, no. 4, pp. 316–344, 2005.
[10] Á. Lédeczi, Á. Bakay, M. Maróti, P. Völgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai, "Composing Domain-Specific Design Environments," *Computer*, vol. 34, no. 11, pp. 44–51, 2001.
[11] D. Saha, R. Mitra, and A. Basu, "Hardware software partitioning using genetic algorithm," in *VLSI Design, 1997. Proceedings., Tenth International Conference on*. IEEE, 1997, pp. 155–160.
[12] Y. Vanrompay, P. Rigole, and Y. Berbers, "Genetic algorithm-based optimization of service composition and deployment," in *Proceedings of the 3rd international workshop on Services integration in pervasive environments*. ACM, 2008, pp. 13–18.
[13] A. Shah, K. An, A. Gokhale, and J. White, "Maximizing service uptime of smartphone-based distributed real-time and embedded systems," in *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2011 14th IEEE International Symposium on*. IEEE, 2011, pp. 3–10.
[14] J. White, B. Dougherty, C. Thompson, and D. C. Schmidt, "Scatterd: Spatial deployment optimization with hybrid heuristic/evolutionary algorithms," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 6, no. 3, p. 18, 2011.

Optimizing the Linear Real-Time Logic Verifier

Albert M. K. Cheng
Department of Computer Science
University of Houston
Houston, TX 77004, U.S.A.
cheng@cs.uh.edu

Stefan Andrei
Department of Computer Science
Lamar University
Beaumont, TX 77710, U.S.A.
stefan.andrei@lamar.edu

Mozahid Haque
Department of Mathematics
University of Houston
Houston, TX 77004, U.S.A.
mahaque@uh.edu

ABSTRACT

In [1], Cheng and Andrei introduce a new method of implementing the verification of an extension of Real-Time Logic, RTL, called Linear Real-Time Logic, LRTL. LTRL allows for dependencies beyond two events and replaces the standard practice of constraint graphs with matrix operations. This paper describes an algorithm for solving the matrix problem using QR decomposition through Householder reflections. The purpose of this paper is to introduce the problem, to introduce and discuss the QR algorithm compared to the previous Gaussian algorithm described in [1] and [2], and to show the implementation and the increased efficiency of the QR algorithm.

General Terms

Algorithms and Verification.

Keywords

Linear Real-Time Logic, LTRL, Real-Time Logic, RTL.

INTRODUCTION

Formal specification and its verification are known to be very important for the development of safe real-time and embedded systems. In RTL, we are given a set of safety specifications and safety assertions, SP and SA, respectively. Then, we show that SP → SA through the logically equivalent form SP Λ ~SA, where ~SA is the negated version of SA. We show the unsatisfiability of the negative logically equivalent form, i.e. ~(SP Λ ~SA). One such logic for describing the specification and the safety assertion is Real-Time Logic (RTL). However, the satisfiability problem for RTL, as well as for other first-order logics, is undecidable. In an effort to find subclasses of RTL having decidable properties, the path real time logic (path RTL) was described in [11]. A typical timing constraint expressed in path RTL is a disjunction of inequalities like $\forall i$ @($e1$, i)− @($e2$, i) ≤ k, that has the meaning: the difference between the time of the i−th occurrence of event $e1$ and the time of the i−th occurrence of event $e2$ is at most k, where the time occurrence, i and k are positive integers. The class of path RTL formulas were successfully used to specify industrial real-time systems, e.g., railroad crossing, the moveable control rods in a reactor, the Boeing 777 Integrated Airplane Information Management System, and the X 38, an autonomous spacecraft designed and built by NASA as a prototype of the International Space Station Crew Return Vehicle.

If the constraints are given in the form of inequalities, we can express this in conjunctive normal form after expressing the events in function form, i.e., a conjunction of inequalities. Then, after Skolemization of quantifiers, we are left with a simple set of inequalities in the form:

$$\sum_{j=1}^{n} a_{1,j} * x_j \leq b_1$$

$$\sum_{j=1}^{n} a_{2,j} * x_j \leq b_2$$

. . . .

$$\sum_{j=1}^{n} a_{n,j} * x_j \leq b_n$$

The simple form is further simplified with the linear algebra notation of **Ax** ≤ **b**, giving rise to Linear Real-Time Logic, abbreviated LRTL [1], [2].

This is a simplified version of the derivation of the problem. This is better and more thoroughly articulated in [1] and [2]. Papers [1] and [2] were successfully used and compared to similar works about specification, verification, and scheduling of real-time embedded systems, such as [7], [9], and [10]. The verification technique described in [1] and [2] involves solving a system of linear equations using the Gaussian elimination method. However, it is known that scaling of equations is a source of numerical errors in Gaussian elimination method. Hence, the numerical instability of Gaussian elimination method can be reduced by doing the so-called pivoting method. Despite that, Gaussian elimination with pivoting may be highly inaccurate for some matrices [5]. In comparison, the Householder's reducing method is unconditionally stable, both in theory and practice [5]. Likewise the Gaussian elimination method, the Householder's reducing method can be easily implemented on a parallel architecture using a parallel algorithm for solving the system of linear equations.

The main contributions of this paper are two-fold:

1. The paper describes an improved algorithm for verification of real-time embedded systems using the stable Householder method of solving systems of equations. The previous algorithm described in [1] and [2] is transformed by changing the Gaussian elimination method with the Householder reducing method when solving the system of linear equations.

2. The paper discusses the implementation of this improved algorithm and compares to the execution time of the previous algorithm. We conclude that an additional 40-50% execution time compared to the implementation from [1] and [2] is worth it given the

*Supported in part by the National Science Foundation under Awards No. 0720856 and No. 1219082.

fact that there are no numerical errors due to the discrepancies between large and small coefficients.

1. The Problem

We recall the specification of the radar station specification (adapted from [3]): "A radar system searches objects of interest in the desired coverage area by repeatedly executing the following steps: (1) scanning/radio signal processing, (2) tracking, and (3) data association/classification. Here, we specify a simplified version of the specification of the tracking step for four objects. The safety assertion states that the computing resources (2 CPUs) can feasibly schedule the four object-tracking tasks, each tracking a distinct object of interest. Each task is fully parallelizable and thus can execute on two CPUs if needed to speed up its execution by a factor of 2. Tasks *T*1, *T*2, *T*3, and *T*4 have respectively computation times *c*1, *c*2, *c*3, and *c*4, all with the same period of *p*." Due to the environmental constraints, the standard (original) specification will change into an "Extreme" SP. The computation times c1,...,c4 and p are c1 = 99,999,800, c2 = 200, c3 = 99,999,900, c4 = 100, and p = 100,000,000. In this way, more objects are being tracked (so more c_i's) and more extreme variations in the values of c_i's. In order to translate the above natural language specification to an extended path RTL formula, we denote by *Tj CPUk* the fact that task *j* is executing in *CPUk*. For any $j \in \{1, ..., 4\}$ and $k \in \{1, 2\}$, we denote by @(↑ *Tj CPUk, i*) and @(↓ *Tj CPUk, i*) the *i*-th occurrence of the starting and the ending time of task *Tj CPUk*, respectively. The above specification can be written in the extended path RTL as *SP*:

@(↓ *T*1 *CPU*1, *i*) + @(↓ *T*1 *CPU*2, *i*) ≤ *c*1+ *c*2+ *c*3+ *c*4
@(↓ *T*2 *CPU*1, *i*) + @(↓ *T*2 *CPU*2, *i*) ≤ *c*2+ *c*3+ *c*4
@(↓ *T*3 *CPU*1, *i*) + @(↓ *T*3 *CPU*2, *i*) ≤ *c*3+ *c*4
@(↓ *T*4 *CPU*1, *i*) + @(↓ *T*4 *CPU*2, *i*) ≤ *c*4

These inequalities form together a system of inequalities $\mathbf{Ax} \leq \mathbf{b}$, where **A** is an *n* x *n* matrix, and **x** and **b** are n-tuple vectors. This will be a two-step process where we first solve for the null space of $\mathbf{A^T}$, i.e., find $\boldsymbol{\lambda}$ such that $\mathbf{A^T}\boldsymbol{\lambda} = \mathbf{0}$ where **0** is the zero vector. Additional constraints on $\boldsymbol{\lambda}$ require that $\boldsymbol{\lambda}$ be a vector consisting of positive real numbers. Then we check whether the dot product of $\boldsymbol{\lambda}$ and **b** is negative, i.e., $\boldsymbol{\lambda}^\mathbf{T}\mathbf{b} < 0$.

We will treat the problem as two separate entities that can be recombined for the desired solution of our original problem— the two entities being solving the null space of an *n* x *n* matrix with so that the vector consists of positive elements and making the dot product negative, motivating some of our reasoning as we go along.

1.1 $A\lambda = 0$

Since our matrix is *n* x *n*, in order for the null space to consist of more than just the trivial **0** vector, our matrix must be singular or rank-deficient. Otherwise, our null space will just consist of the zero vector, an undesirable result given our conditions on $\boldsymbol{\lambda}$. So let us suppose the null space is non-empty.

One practice involves the familiar Gaussian elimination where we reduce the matrix **A** and decompose it such that $\mathbf{A} = \mathbf{LU}$, where **L** is a lower triangular matrix consisting of the multipliers generated by the diagonal elements of **A**. A series of elimination steps is performed until **A** is transformed to **U**, where **U** is an upper triangular matrix which allows for relatively easy back substitution to solve for $\boldsymbol{\lambda}$, in other words $\mathbf{A}\boldsymbol{\lambda} = \mathbf{LU}\boldsymbol{\lambda} = \mathbf{0}$. A foreseeable problem with this method is the lack of pivoting. If one of the pivot elements is a zero, then it is not possible to eliminate elements below that pivot.

Now, let us consider the following example from Forsythe and Moler(1967):

$$\begin{pmatrix} 0.0001 & 1 \\ 1 & 1 \end{pmatrix}$$

When we attempt to apply Gaussian elimination with three-point precision, we calculate the multiplier as usual $m_{21} = \frac{-1}{0.0001} = -10^4$. So **U** and **L** are

$$U = \begin{pmatrix} 0.0001 & 1 \\ 0 & 0.0001 \end{pmatrix} \quad L = \begin{pmatrix} 1 & 0 \\ 10000 & 1 \end{pmatrix}$$

Computing **LU** with three-point precision gives us

$$LU = \begin{pmatrix} 0.0001 & 1 \\ 0 & 1 \end{pmatrix}$$

This does not result in retrieving our original matrix; something was lost in between. One may argue it is the result of the precision chosen. However, this example's purpose was to illustrate the shortcomings of a computer's precision, which is finite. Rounding errors due to machine precision led naturally to a desire for machine-independent algorithms.

To alleviate the potential loss in information after reduction, a method of pivoting is introduced to realign the elements in a given matrix so that the pivot elements are relatively large comparatively. This reduces the likelihood of generating large multiplies and a more stable algorithm overall. The three types of pivoting are column pivoting, row pivoting, and complete pivoting. Column pivoting involves picking out the largest element in a column and exchanging rows, requiring a reordering of the elements in **b**. Row pivoting involves picking out the largest element in a row and exchanging columns, requiring a reordering of the elements in $\boldsymbol{\lambda}$. Complete pivoting involves a combination of both. Complete pivoting is far more efficient for very large matrices, but for our purposes, we will stick to just column pivoting. An algorithm can easily be adjusted for the other forms of pivoting.

In lieu of Gaussian elimination without pivoting, we present the algorithm for Gaussian elimination with column pivoting for an n x n matrix from various sources. For i, $1 \leq i \leq n-1$,

1. (Find the max in the i^{th} column) Find and store the row index, m_i, of the max element in each column $\mathbf{a}_i$, where $\mathbf{a}_i$ are the column vectors starting with the element on the diagonal to the end of the column, i.e. the row index below goes from $i < m_i \leq n-1$. If the max happens to be zero, then stop because that column has been zeroed out already.
2. (Row exchange, pivoting) Use the row index m_i and exchange that row with the i^{th} row.
3. (Multiplier formulation) Use the new pivot to formulate multipliers for Gaussian elimination.
4. (Reduction) Use the multipliers to reduce the matrix and zero out the elements.

This algorithm uses about $\frac{n^3}{3}$ flops and $O(n^2)$ comparisons. Then we would use back substitution to calculate the null space after this reduction.

However, another issue arises when using any iterative matrix transformation. The elements within the matrix may get larger/smaller compared to the original elements with each iteration. This will lead back to the scenario by Forsythe and Moler of potential rounding errors. A more efficient method involves QR decomposition through Householder transformations and reduction to a Hessenberg matrix.

2. QR Decomposition Using Householder Reflections

A Householder matrix preserves the length of a vector, which is useful in transforming a given vector to some multiple of a basis vector. Matrices, in general, transform a vector to another vector; in this case, the transformation will function similarly to a Gaussian elimination matrix to zero out elements in a vector.

2.1 Background

A Householder matrix, in the context of linear algebra, is an orthogonal matrix. This is a special matrix where its transpose is its inverse. Computation of matrix problems become less complex and more time-efficient when one can use special structures of a given matrix. In this case, we can perform a method of zeroing out a vector with a Householder matrix.

In other words, given some nonzero column vector, construct the Householder matrix $\mathbf{H} = \mathbf{I} - 2 * \frac{\mathbf{u}*\mathbf{u}^{\mathrm{T}}}{\mathbf{u}^{\mathrm{T}}*\mathbf{u}}$ where $u = v - \mathrm{sign}(v_1) * |v|_2 * e_1$. Sign refers to whether v_1 is positive or negative; $|v|_2$ is the Euclidean norm; and e_1 is just the identity column vector (1,0,...,0). If v_1 is 0, then we may pick the sign freely. Then we have that

$$Hv = \begin{pmatrix} \mathrm{sign}(v_1) * c \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

We get a zeroed out column vector. This process is numerically stable. Due to the increased stability with such a process that zeroes out a column vector, one can imagine applying this to a matrix and zeroing out elements to get an upper triangular matrix, which is what QR factorization is. Moreover, the Householder reduction method has the same time complexity as the Gauss elimination method, that is, $O(n^3)$.

Before we proceed further in laying out this more efficient algorithm, let us reexamine our original goal. We wish to find the null space of a given matrix. For our null space to contain a vector other than the trivial **0** vector, our matrix must be rank deficient. Assume that our matrix is rank deficient. Then after reduction through various techniques to an upper triangular matrix, we will never have a last row with a single nonzero element. At the very least, the last row will be a row of zeroes because our matrix was originally assumed to be rank deficient. With that in mind, it would be inefficient to reduce the entire matrix using Householder matrices on the diagonal (QR factorization). Instead, it would be more cost effective to reduce the matrix to an upper Hessenberg matrix. An upper Hessenberg

matrix has the form $H = \begin{pmatrix} * & * & * & \cdots & * & * & * \\ * & * & * & \cdots & * & * & * \\ 0 & * & * & \cdots & * & * & * \\ 0 & 0 & * & \cdots & * & * & * \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & * & * \end{pmatrix}$.

For an nxn matrix, it has $\frac{(n-2)(n-1)}{2}$ zeroes as opposed to $\frac{n(n-1)}{2}$ zeros for the other normal reductions. Having less zeroes means less transformation of the original matrix reducing the total amount of work and potential loss of information due to rounding errors. To illustrate reduction to an upper Hessenberg matrix rather than below the diagonal, consider the following matrix:

$$\begin{pmatrix} 1 & 2 & 1 \\ 1 & 0 & 3 \\ 1 & 1 & 2 \end{pmatrix}$$

If we reduce using Gaussian elimination below the main diagonal, we reduce the matrix to

$$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 2 & 3 \\ 0 & 0 & 0 \end{pmatrix}$$

in three steps- the lower corner elements. Then we would solve for the null space. However, consider eliminating just the lower left corner element to get

$$\begin{pmatrix} 1 & 2 & 1 \\ 1 & 0 & 3 \\ 0 & 1 & 1 \end{pmatrix}$$

This is just one step of zeroing out as opposed to three. We can immediately start back substituting after that since we are solving for the null space. We can always just pick values for a reduced matrix when solving for the null space in this manner:

$\begin{pmatrix} * & * & * \\ * & * & * \\ 0 & a & b \end{pmatrix}$ – if a and b are opposite signs, let $x_2 = b$ and $x_3 = a$ and if they are the same sign, then we would just make one of them negative. This implementation with QR decomposition would reduce some of the work.

2.2 Proposed Algorithm

Let **A** be a rank deficient nxn matrix so that the null space contains more than the trivial zero vector.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

$$\mathbf{u} = \mathbf{a}_i - \mathrm{sign}(a_1) * |\mathbf{a}_i|_2 * e_1 \qquad \text{(eq. 1)}$$

$$\mathbf{H} = \mathbf{I} - 2 * \frac{\mathbf{u}*\mathbf{u}^{\mathrm{T}}}{\mathbf{u}^{\mathrm{T}}*\mathbf{u}} \qquad \text{(eq. 2)}$$

For i, $1 \le i \le n-2$,

1. Define $\mathbf{a}_i = \begin{pmatrix} a_{i+1,i} \\ a_{i+2,i} \\ \vdots \\ a_{n,i} \end{pmatrix}$ from matrix **A**.

2. Apply (eq. 1) and (eq. 2) to the vector in step 1.

3. Define $\mathbf{H}_i = \begin{pmatrix} \mathbf{I}_{i-1} & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \mathbf{H} & \\ 0 & & & \end{pmatrix}$.

4. Update the matrix by $\mathbf{A}_i = \mathbf{H}_i\mathbf{A}_{i-1}$ where $\mathbf{A}_0$ is the original matrix.

In step 3, $\mathbf{I}_{i-1}$ means the identity matrix up to the index element; then the remaining portion of the matrix would be the **H** matrix from step 2. If the resultant matrix happens to be rank deficient by two, there may be a need to reduce further than just Hessenberg. This could be done through Gauss since it would just be one more reduction along the diagonal without too much concern about stability. The last step would be to solve for the null space of the final matrix $\mathbf{A}_{n-2}$ through back substitution and multiply by the vector **b** to check for negativity.

3. Conclusions

For sufficiently small enough matrices, the proposed methods in [1] and [2] using Gaussian elimination with pivoting is stable enough for practice. However, QR decomposition through Householder reduction has increased numerical stability for large matrices compared to other linear equations methods. The flop counts are relatively the same, but their respective numerical stability concerning matrix size and large differences between coefficients within the matrix sets Householder reduction apart from other methods.

Additionally, we could not form the matrix **H** at all and just do a few vector multiplications with the updated matrix **A**'s with the following identity:

$$(\mathbf{I} - 2 * \frac{\mathbf{u} * \mathbf{u}^T}{\mathbf{u}^T * \mathbf{u}})\,\mathbf{A} = \mathbf{A} - \frac{\mathbf{u} * \mathbf{u}^T}{\mathbf{u}^T * \mathbf{u}}\,\mathbf{A}.$$

This allows us to carry out less flops and better implement in practice.

Currently, we hope to explore additional methods that may be more effective without compromising the overall goal. We hope to find methods that may exploit any sparsity patterns that generally may arise from a given set of safety specifications (SP) and retaining the sparsity as we go through reductions like in the Gaussian elimination method.

3.1 Stability

Average case studies show that Gaussian elimination with partial pivoting have a growth factor $O(2^n)$ whereas QR decomposition has a growth factor of about $O(n^2)$. This provides a more numerically stable case for QR. However, in practice, Gaussian elimination has been shown to be fairly stable in practice due to non-exotic nature of a typical matrix that may come up in practice. QR decomposition still provides more stability for sufficiently large matrices.

3.2 Additional Sources of Stability

Earlier we mentioned the importance of pivoting to illustrate the larger concern of the elements in a given matrix being blown out due to the presence of large elements relative to the other elements after carrying out transformations. So we wish to address this issue by introducing a form of scaling of the elements in a given vector that retains the general information but reduces the impact of larger elements. This increases the number of the elementary arithmetic operations, but increases stability of the Householder reduction method when compared with the Gauss elimination method. We search for the largest element in the vector we are about to transform. Then we scale the other elements relative to the largest element in a vector. We take the ratio with the largest element, call it *m*, so that all of our elements in a vector, say **v**, becomes scaled by $(1/m)\mathbf{v}$. This may or may not be useful in this context so we eliminated it from the proposed method. The simple method of picking values for a and b at the end of Section 2.1 could be harmful depending on the nature of their largeness. However, we could additionally implement a pivoting schema for QR decomposition similar to Gaussian elimination. Additionally, the scaling method laid out earlier would reduce such potential harm. Implementation of this can be easily remedied. We note that there are further rounding error strategies beyond this scaling one that may further enhance stability. In the full version of this paper, we will compare the performance of the proposed approach with existing implementations as well as with competing verification tools.

Acknowledgments

Thanks go to Dylan Thompson for help in implementing the new analysis approach.

References

[1] Andrei, S. and Cheng, A. M. K. 2007. *Verifying Linear Real-Time Logic Specifications*. 28th IEEE International Real-Time Systems Symposium (RTSS), Tuscon, AZ, 2007.

[2] Andrei, S. and Cheng A. M. K.: Efficient Verification and Optimization of Real-Time Logic Specified Systems, IEEE Transactions on Computers, Vol. 58, No. 12, pp. 1640-1653, 2009.

[3] Andrei, S., Cheng, A. M. K.: Faster Verification of RTL-Specified Systems via Decomposition and Constraint Extension. 27th IEEE Real-Time Systems Symposium (RTSS), Rio de Janeiro, Brazil, December 5-8, 2006.

[4] Datta, B. N. 1995. *Numerical Linear Algebra and Applications*. Pacific Grove, CA. Brooks/Cole Publishing Company.

[5] Forsythe, G. E. and Moler, C. B. 1967. *Computer Solutions of Linear Algebraic Systems*. Englewood Cliffs, NJ. Prentice Hall.

[6] Per Brinch Hansen. 1992. Householder Reduction of Linear Equations, *ACM Computing Surveys*, Vol. 24, No. 2, June 1992, pp. 185-194.

[7] McDonough, J. M. *Lectures in Basic Computation Numerical Analysis*. 2007. Lexington, KY. University of Kentucky.

[8] Jim Ras and Albert M. K. Cheng. 2010. *On the Toyota's Throttle Control Problem*. IEEE/ACM Int'l Conf. on Green Computing and Communications & Int'l Conf. on Cyber, Physical and Social Computing (GREENCOM-CPSCOM '10), Washington, DC, USA, pp. 889-894.

[9] Spedicato, E. *Computer Algorithms for Solving Linear Algebraic Equations*. 1991. Berlin, Germany. Springer-Verlag.

[10] Terry Tidwell, Robert Glaubius, Christopher D. Gill, William D. Smart: Scheduling for Reliable Execution in Autonomic Systems. Proceedings of the 5th International Conference on Autonomic and Trusted Computer (ATC), 2010, LNCS 5060, pp. 149-161.

[10] Zhong Liang Pan, Ling Chen: A New Verification Method of Digital Circuits Based on Cone-Oriented Partitioning and Decision Diagrams. Applied Mechanics and Materials, Vol. 29-32 (Applied Mechanics and Mechanical Engineering), Trans. Tech. Publications, Switzerland, 2010, pp. 1040-1045.

[11] F. Jahanian and A. Mok. A graph-theoretic approach for timing analysis and its implementation. *IEEE Transactions on Computers*, C-36(8):961–975, 1987.

Predictive Scheduling for Spatial-dependent Tasks in Wireless Sensor Networks

Hua Huang, Shan Lin, Anwar Mamat and Jie Wu
Department of Computer and Information Sciences
Temple University
Philadelphia, PA 19122
{*hua.huang, shan.lin, anwar, jiewu*}*@temple.edu*

***Abstract*—Recent advances in wireless energy transfer technology boost renewable sensor networks. To sustain the operation of a sensor network, a mobile charger is used to recharge each sensor node before its battery runs out of power. Consider each node's recharge as a task with a real-time due time, the mobile charger needs to dynamically schedule these tasks. This scheduling problem is very challenging, since nodes' close due time can conflict with its long travel distance. Existing solutions to this problem usually assume fixed paths generated by the travelling salesman or Hamilton cycle algorithms. These solutions suffer from high deadline miss ratios when nodes far away with the charger have close due times. In this paper, we investigate maximum response ratio based scheduling algorithms that dynamically select the path of chargers for higher network coverage ratio. Since each task's execution time depends on mobile charger's current location, our algorithms model the spatial dependency among proximate tasks, and predict the impact of one charging task to the others. With extensive trace-driven analysis, our algorithms significantly outperform existing algorithms.**

I. Introduction

Sensor networks are deployed for various military surveillance [1], scientific exploration [2], and first responder applications [3]. Such applications may use visual or acoustic sensors, which have high energy consumption rates. To sustain long-term system operations, sensor nodes need to be recharged periodically after their deployments. With recent advances of wireless energy transfer technology, a mobile wireless charger can travel through the network to recharge sensor nodes [4], [5]. In small scale systems with low energy consumption rates, the recharge schedule is trivial. The charger can simply traverse the network to recharge every node one by one on a fixed path. However, in large scale networks with high energy consumption rates, the travelling time of mobile charger is comparable to the lifetime of sensor nodes. A node can run out of energy if the charger fails to recharge it in time. It is very challenging to find a recharge schedule that keeps every node alive while minimizing the travelling cost.

The intrinsic spatiotemporal constraints make this dynamic mobile charge scheduling problem unique and challenging. The solution to this problem clearly depends on the ratio between travelling time of the mobile charger and the lifetime of sensor nodes. Take one extreme, say the ratio is close to zero (travelling time is much shorter than sensors' lifetime), the mobile charge scheduling algorithm boils down to the real-time scheduling problem on a single machine. This can be solved by classic algorithms such as Earliest Deadline First(EDF) algorithm. Take another extreme, say the ratio is close to infinity(travelling time is much longer than sensors' lifetime), mobile charger basically needs to select the most efficient route to pass every node. This becomes the travelling salesman problem, which is well-known to be NP-hard. When the ratio is close to one, this problem is not studied before. There lacks efficient solutions: classic scheduling algorithms [8] focus on task deadlines at a centralized location, while computational geometry algorithms [9] mainly seek efficient routes in multi-dimensional spaces regardless of deadlines. There are a few related works [6], [7] deal with the mobility scheduling in the context of sensor networks. Different from these works, we focuses on scheduling algorithm designs dealing with spatiotemporal dynamic tasks.

To address this problem, we take each node's recharge request as a task of a soft real-time due time. We note that the due time of each task is independent of each other, since the energy consumption rate of each node is determined by its own workloads. However, these tasks are dependent in space. Given the same due time, recharging a node with high spatial proximity requires much less travelling overhead than recharging a node far away. To characterize this special feature of mobile charge, we introduce the spatial dependency task model. This model quantifies the impact of each recharge on the other tasks. The spatial dependency model allows tasks close to travelling path to have higher priorities than the other tasks, but it is not enough to avoid local optimal decisions. This is partially due to the lack of global information in a distributed large scale networks. Therefore, we further design predictive heuristics to look ahead with rough grained global task estimations. Our solution demonstrates a good tradeoff between scheduling coverage and cost. We evaluate our solutions in extensive simulations with energy consumption traces obtained from real sensor network deployments. The simulation results show that significantly higher coverage ratio with little travelling overhead can be achieved.

II. Task and System Models

We consider a sensor network of n nodes deployed in a two-dimensional space. These sensor nodes have independent energy consumption rates. A Mobile Charger (MC) is a vehicle equipped with a wireless power recharge device. It travels through the network to deliver energy to the sensor nodes one by one.

1) Task Model: the recharge requests of a sensor nodes are modelled as a task. A task τ_i is represented by three parameters

$\{p_i, t_i^w, d_i\}$. Since these parameters are dependent on time, the MC has to regenerate the task list each time when it is to make recharge schedule decisions. These parameters are described as below:

Processing time p_i is modelled as the travelling time for the mobile charger to move to next target. If we use v to represent MC's speed and l_{next} to represent the distance between current and target nodes, then p_i can be calculated as bellowed.

$$p_i = \frac{l_{next}}{v} \tag{1}$$

Waiting time t_i^w is modelled as the amount of time the node has been waiting since its last recharge.

Soft due time d_i is modelled as the amount of time remained before the corresponding node's energy depletes. In some cases, tasks are already past due when the task list is generated. So d_i is defined to be negative under these conditions. If we denote the node's maximum energy storage by e^{max} and energy consumption rate by r_i, then the due time of a task can be calculated as below:

$$d_i = e^{max}/r_i - t_i^w \tag{2}$$

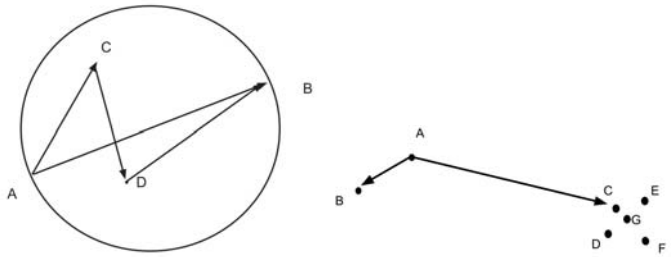
When $t_i^w > e^{max}/r_i$, it means that the node has been waiting for too long that it has already run out of energy.

2) Spatial Task Dependency: Different from traditional task scheduling problems [8], [10], [11], the execution of one task can affect the processing time of all other tasks in our Dynamic Mobile Charger scheduling problem. This is because when the MC begins executing one task, it moves and changes its relative locations to all other nodes. This will in turn affect all their processing time by equation 1

An example is shown in Figure 1a. Assume MC is located at node A. Then the processing time of the task to recharge node B is $p(B) = d(\overline{AB})/v$. But if MC is located at node C, then the processing time of the same task is $p(B)' = d(\overline{CB})/v$. In this case, $p(B) > p(B)'$. In other words, the time it takes to recharge node B depends on the MC's current location This indicates that potential performance gain could be attained by exploring the sensor nodes' spatial dependencies. We discuss two motivating examples as followed.

Nodes near MC's path: In Figure 1a (a), suppose the MC's path is from node A to node B. Processing time of τ_B is $p(B) = d(\overline{AB})/v$. If B's due time is $d_B > p(B)$, we can take advantage of this laxity and allow MC to recharge other nodes before B. If $p(C) + p(D) + p(B)' = d(\overline{ACDB})/v < d_B$, then MC can recharge node C, D before B without missing B's due time. The additional cost for this new scheduling can sometimes be small if C and D are located near path $\overline{AB}$. The panelty introduced for τ_B is $p(C) + p(D) + p(B)' - p(B)$.

Node clusters: Nodes with approaching due times and high proximity form clusters to be recharged. As illustrated in Figure 1b, say MC is located at node A, the processing time of node B is shorter than the other nodes ($p(b) < p(d) \approx p(e) \approx p(f) \approx p(g) \approx p(h)$). However, if MC recharges B first, other nodes will die before being recharged. A solution for lower due time miss ratio is to first charge the node clusters (C, D, E, F, G) with a higher density of approaching due times.



(a) Nodes near path (b) Spatial Cluster

Fig. 1: Task Spatial Dependency

3) Model Metrics: Based on this task model, we can define the following metrics to evaluate any scheduling algorithms.

Coverage: If the energy of a sensor node runs to 0, it will stop functioning. We define the Coverage of the network to be the percentage of sensor nodes being alive at the moment. It can be defined by the following equation:

$$Coverage = \frac{\text{Number of functioning nodes}}{\text{Number of all nodes}} \tag{3}$$

MC Cost: $Cost$ denotes the energy overhead for the MC to recharge the sensor network. In this paper, we consider only the travelling cost of MC. Since MC is travelling at a constant speed, the cost is linearly proportional to the travelling time:

$$Cost = \sum_{\text{all charged nodes}} K \times p(i) \tag{4}$$

where K is a constant denoting the energy consumption per unit time.

III. Algorithm

In this section, we propose several algorithms for MC scheduling. We assume once MC starts moving forward a node, scheduler does not schedule tasks until the target node is fully charged and energy conditions of all other nodes are updated. Our goal is to provide reliable network such that the coverage ratio is maximized, while maintaining fairness between nodes and between regions.

A. Minimum Weight First

In Minimum Weight First(MWF), we seek for a tradeoff between processing time and due time. Therefore, we employ the parameter α to represent their relative importance.

$$Priority_i = \alpha p_i + (1 - \alpha) max(0, d_i) \tag{5}$$

where p_i is the processing time defined in equation 1. It represents the cost of executing the task. d_i is the node's soft due time by equation 2. The task with minimum $Priority_i$ will be selected as the next target. When $\alpha = 1$, the problem becomes the Greedy TSP. When $\alpha = 0$, only deadline affects the schedule.

B. Maximum Response ratio First

Maximum Response ratio First(MRF) scheduler defines the priority of each task to be dependent on both its processing time and waiting time. The longer tasks wait, the higher priority they gain. This will prevent infinite postponement (process starvation) [8]. Intuitively, the higher energy consumption rate a node has, the less time it can afford to wait. Therefore we multiply the energy consumption rate r_i with the waiting time t_i^w. MRF's priority is defined as followed:

$$Priority_i = \frac{t_i^w \times r_i + W_d max(0, -d_i)}{p_i} \quad (6)$$

The term $max(0, -d_i)$ will be non-zero only when a task is past due. We assign the weight W_d to this term to reflect the additional penalty of lateness.

C. Spatial Laxity based Heuristics

As described in section II, spatial dependency of nodes exists. Based on these observations, we propose two heuristics: Spatial Laxity Filling(SLF) and Spatial Laxity Clustering(SLC). SLF searches nodes near MC's travelling path. SLC partitions the sensor network into spatial clusters and computes each node's priority based on both its own and its cluster's energy conditions.

1) Spatial Laxity Filling: As illustrated in figure 1a, Assume MC is located at node A. The Spatial Laxity Filling(SLF) scheduler firstly selects the node using MRF, say B. Then the scheduler will check if there are any nodes located inside the circle whose diameter being line $\overline{AB}$. If so, SLF will select the node that lies closest to line $\overline{AB}$ to be recharged before B.

2) Spatial Laxity Clustering: Spatial Laxity Clustering(SLC) divides the network into different clusters and takes into account the estimation of each cluster's urgency when scheduling. SLC firstly divides the node space into m non-intersecting clusters. SLC defines the priority of each cluster using the following equation:

$$Priority^c = \frac{\sum_{node\ in\ cluster} t_i^w \times r_i}{p^c} \quad (7)$$

where t_i^w is the waiting time and r_i is the energy consumption rate of each node. The term $\sum_{node\ in\ cluster} t_i^w \times r_i$ is an estimation of the cluster's urgency. p^c is modelled as the travelling time for MC to travel from its current location to the geometric centroid of the node cluster.

SLC will select the next cluster with maximum $Priority^c$. After entering a cluster, MC will be scheduled using MRF to recharge nodes within this cluster. After the urgency estimation of the cluster falls below a threshold, the charger will leave to next cluster. Using this algorithm, better fairness among different clusters can be achieved.

IV. Evaluation

We have implemented a simulation framework to evaluate the scheduling algorithms proposed in this paper. In the simulation, the sensor network consists of 15×15 nodes deployed in a grid manner. We assume charger has the global energy information of all the nodes. A node's recharge is assume to be finished at the moment MC moves to it. We will present the results of MRF, SLF, SLC and MWF with $\alpha = 0.2, 1$ in this section.

A. Coverage

Firstly, we compare the performance of different mobile charger algorithms in terms of network $Coverage$ defined in equation 3. In each experiment, the charger totally recharges 10000 times before it stops. Then we compute the network's average coverage ratio during the experiment. For each algorithm, we repeat this experiment 50 times and compute the average of these two statistics. The results are shown as below.

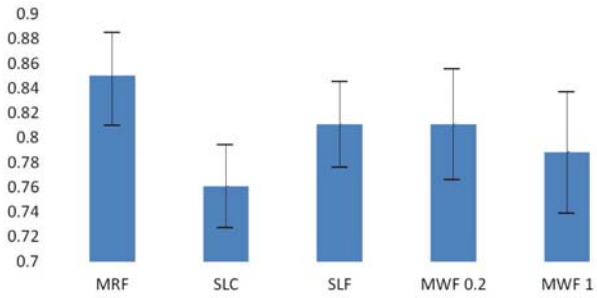


Fig. 2: Coverage

From figure 2 we can see that MRF has best performance. It is due to its ability to achieve a balance between waiting time and processing time in scheduling. On the other hand, SLC has a lower performance. The reason is that a large travelling cost can occur during cluster switching.

When MWF algorithm has the parameter $\alpha = 0.2$, it has a better performance than when $\alpha = 1$. However, as can be seen in later analysis, this is at the expense of higher recharging overhead.

B. Due Time

Each time a sensor node is being recharged, we record that node's Due Time defined in equation 2. The Cumulative Distribution Functions(CDF) of the Due time are plotted as below:

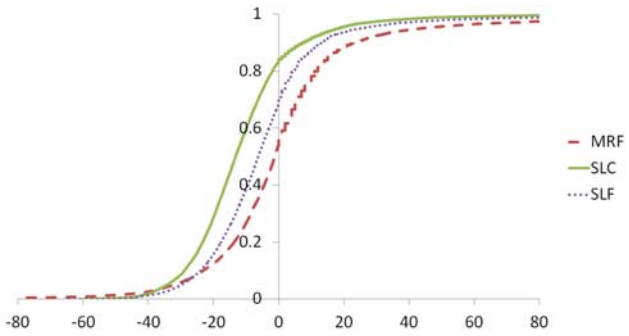


Fig. 3: Due Time

From figure 3 we can see that MRF is able to maintain a smaller number of dead nodes. This is because MRF focuses more on nodes with low energy. On the other hand, although SLF can ensure next task's due time will not be missed, a larger

number of dead nodes still occur. This is because inserting nearby nodes can increase the waiting time of nodes lying far away.

C. Fairness

In order to evaluate the spatial fairness of these scheduling algorithms, we divide the sensor network equally into 3×3 clusters with rectangle shape. Each cluster has 5×5 nodes. The coverage ratio is computed for each cluster using the following equation

$$Coverage_i = \frac{\text{Number of functioning nodes in } Cluster_i}{\text{Number of all nodes in } Cluster_i} \tag{8}$$

Then we can compute the standard deviations of $Coverage_i$ between different clusters for each algorithm. The results are shown as below.

Fig. 4: Fairness

This figure shows that SLC has the lowest standard deviation, which means the coverage ratios between different clusters are similar. This is due to the fact that SLC can adjust its scheduling based on each cluster's need. On the other hand, MRF has high standard deviation. This indicates that MRF achieves higher coverage ratio at the expense of spatial fairness. The fairness of MWF with $\alpha = 1$ is the worst. This is because this scheduler favors too much about nearby nodes, thus produces unfairness to nodes far away.

D. Cost

We sum up all the travelling distance of the MC for each experiment. Then we normalize these distance by dividing them by the travelling cost of MWF with $\alpha = 1$. This is because MWF with $\alpha = 1$ is a classical greedy solution to Travelling Salesman Problem. Besides, it achieves minimum travelling cost in all the algorithms we designed.

As shown in 5, we can see that SLC has a large travelling cost, due to its long distance travel when changing clusters. SLF achieves a lower cost than MRF because it selects to recharge nodes which are located close to the recharging route. This proves to be efficient in terms of travelling cost.

To summarize, we can see that MRF can achieve a higher coverage ratio. The advantage is more significant when the recharge workload is heavy. When we hope to minimize the travelling cost of MC, we can choose SLF that can reduce

Fig. 5: Cost

travelling cost, while not incurring serious performance degradation. If the geometric condition of the sensor network is so complex that energy consumption rates of nodes from different regions differ greatly, we can choose SLC scheduler so that fairness between different regions can be ensured automatically.

REFERENCES

[1] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, *et al.*, "Vigilnet: An integrated sensor network system for energy-efficient surveillance," *ACM Transactions on Sensor Networks*, 2006.

[2] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, *et al.*, "Envirotrack: Towards an environmental computing paradigm for distributed sensor networks," in *Distributed Computing Systems, 2004.*

[3] H. Liu, J. Li, Z. Xie, S. Lin, K. Whitehouse, J. A. Stankovic, and D. Siu, "Automatic and robust breadcrumb system deployment for indoor firefighter applications," ser. MobiSys '10.

[4] R. Sugihara and R. K. Gupta, "Speed control and scheduling of data mules in sensor networks," *ACM Transactions on Sensor Networks*, 2010.

[5] S. Zhang, J. Wu, and S. Lu, "Collaborative mobile charging for sensor networks," in *Proc. of the 9th IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, 2012.

[6] G. Xing, J. Wang, K. Shen, Q. Huang, X. Jia, and H. C. So, "Mobility-assisted spatiotemporal detection in wireless sensor networks," in *28th IEEE International Conference on Distributed Computing Systems*, 2008.

[7] L. Luo, C. Huang, T. Abdelzaher, and J. Stankovic, "Envirostore: A cooperative storage system for disconnected operation in sensor networks," in *26th Annual IEEE Conference on. Computer Communications*, 2007.

[8] P. Brucker, *Scheduling Algorithms*, 2004.

[9] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, 1992.

[10] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems," *Real-Time Systems*, 1989.

[11] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, "Task scheduling strategies for workflow-based applications in grids," in *Cluster Computing and the Grid, 2005*.

[12] A. K. Wong and A. M. Goscinski, "The impact of under-estimated length of jobs on easy-backfill scheduling," in *Parallel, Distributed and Network-Based Processing*, 2008.

QoS Differentiation in IEEE 802.11 Wireless Local Area Networks for Real-Time Control

Guosong Tian and Yu-Chu Tian
School of Electrical Engineering and Computer Science
Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia
Email: y.tian@qut.edu.au

***Abstract*—IEEE 802.11 based wireless local area networks (WLANs) are being increasingly deployed for soft real-time control applications. However, they do not provide quality-of-service (QoS) differentiation to meet the requirements of periodic real-time traffic flows, a unique feature of real-time control systems. This problem becomes evident particularly when the network is under congested conditions. Addressing this problem, a media access control (MAC) scheme, QoS-dif, is proposed in this paper to enable QoS differentiation in IEEE 802.11 networks for different types of periodic real-time traffic flows. It extends the IEEE 802.11e Enhanced Distributed Channel Access (EDCA) by introducing a QoS differentiation method to deal with different types of periodic traffic that have different QoS requirements for real-time control applications. The effectiveness of the proposed QoS-dif scheme is demonstrated through comparisons with the IEEE 802.11e EDCA mechanism.**

I. Introduction

As the most widely-used wireless networks today, the IEEE 802.11 based wireless local area networks (WLANs) are increasingly deployed in soft real-time control systems. This is due to the increasing demand of connecting components that are unreachable with a cable in industrial environments [1], [2], [3]. Other benefits of the IEEE 802.11 WLANs include mobility support, relatively high transmission speed, low cost and the close resemblance with conventional Ethernet.

Real-time constraints and reliability are two key issues for wireless real-time control applications. To obtain satisfactory quasi-real-time behaviour, quality-of-service (QoS) differentiation is important, particularly in those real-time applications with several different types of data traffic. This has motivated the design of the IEEE 802.11e Enhanced Distributed Control Access (EDCA) for multi-media applications with video and audio traffic over IEEE 802.11 WLANs. However, the traffic behaviours of distributed real-time control systems are quite different from those in multi-media applications [3]. This leads to difficulties in applying the IEEE 802.11e EDCA directly in real-time control systems [4].

The traffic behaviours of networked real-time control systems have been investigated from different perspectives [3], [5]. A unique feature of such traffic behaviours is the periodic traffic pattern. Periodic traffic flows with real-time requirements are expected to have much larger required throughput than other network traffic [6]. In this context, network resources may be mostly utilized by the periodic traffic. The periods of the periodic traffic are typically known in advance under normal conditions [3]. Moreover, periodic data packets are normally fixed in size and are typically short, e.g., a few hundreds or even tens of bytes. In comparison with general computer networks, a networked real-time control system has a relatively smaller number of interconnected nodes (devices), e.g., a few tens or less. The majority of the periodic data packets mostly have deadline requirements. If a periodic packet is transmitted with a delay beyond its deadline, its information becomes out-of-date [3]. In many cases, a periodic packet is dropped if it cannot be delivered to the destination node before its deadline. As in many research papers, this paper also assumes that the deadline of a packet is the same as the period of the periodic traffic.

The QoS differentiation mechanisms for the IEEE 802.11 WLANs, such as the EDCA, have been investigated in real-time control environments. Cena *et al.* employed the four traffic classes (TCs) defined in the IEEE 802.11e EDCA to represent four different types of communications usually found in real-time control applications [6]. The EDCA is able to provide an acceptable real-time QoS when the 802.11 WLAN is under low traffic load conditions. However, if this idea is employed, all periodic traffic flows investigated in [5], [6] would have to be classified into a single TC although they have different QoS performance thresholds (deadlines). Therefore, it does not fulfil the real-time requirements of networked real-time control systems.

To achieve QoS differentiation for periodic traffic with different deadline requirements, a medium access control (MAC) scheme, which we refer to as QoS-dif, is proposed in this paper. It extends the IEEE 802.11e EDCA, and designs a new backoff scheme to replace the IEEE 802.11 binary exponential backoff (BEB) algorithm. In addition, different retry limits as a basic MAC parameter are assigned to periodic traffic flows in terms of their deadline requirements.

II. Background of IEEE 802.11 EDCA

In the IEEE 802.11 standard, a random MAC mechanism with an exponential backoff algorithm is adopted for sharing the network medium. Specifically, if the medium is idle, the node transmits its packet. Otherwise, it postpones its transmission until the medium is sensed free for a time interval that is the sum of an arbitration inter-frame space (AIFS)

and a randomly selected backoff interval. It is permitted to retransmit its packet after the time interval has elapsed for the postponed packet transmission. Therefore, a packet may experience a long and unpredictable delay during the backoff process, particularly when the network is congested.

To improve the MAC's ability to serve and interact with higher level QoS mechanisms, the 802.11e task group has focused on providing differentiated QoS to individual traffic classes (TCs). In particular, the EDCA uses the concept of priority to alter the existing MAC scheme. During initialization, the EDCA assigns static MAC parameters for each of the TCs, such as AIFS and the contention window (CW) range ($CW_{min} \leq CW \leq CW_{max}$). With these parameters, the MAC protocol provides different QoS to each TC. The IEEE 802.11e EDCA parameters are shown in Table I, where AC stands for access category and $AIFSN$ denotes the arbitration inter-frame space number.

TABLE I
IEEE 802.11E EDCA PARAMETER SET [7]

AC	CW_{min}-1	CW_{max}-1	$AIFSN$
AC_VO	3	7	2
AC_VI	7	31	2
AC_BE	31	1023	3
AC_BK	31	1023	7

It is readily realized that the EDCA parameters do not accommodate the deadline requirements of the periodic traffic. In real-time control systems, periodic traffic flows have to be assigned to a single and high priority AC although they have different deadlines [1]. The lack of intra-AC deadline differentiation for periodic traffic may lead to performance deterioration of the real-time control, particularly when the network is under congested conditions. For example, periodic packets with the long deadline may be dropped because of the small retry limit although their deadline is far reached while periodic packets with the small deadline may already miss their deadline. This motivates the research of this paper for a QoS-dif MAC scheme.

III. A QoS Differentiation Enabled MAC Scheme

As described in the previous section, the IEEE 802.11e EDCA supports real-time QoS differentiation, and enables to offer better real-time performance to high priority stations than low priority ones even if the network is under congested condition. However, the real-time performance of high priority stations is not guaranteed. For example, when the network is under high contention, the backoff delay for high priority stations may be very long, even beyond the deadline because of the impact of low priority stations although the delay for high priority stations is much smaller than that for low priority stations.

A significant drawback of the EDCA is observed when there are a number of high priority and low priority stations in a distributed real-time control scenario. The MAC parameters including the contention window CW for the EDCA are pre-defined. It is difficult to configure appropriate values of CW for low priority stations. If low priority periodic traffic is assigned with a small CW, a high collision rate between the low priority periodic traffic flow and high priority ones will appear, and consequently the real-time requirements of high priority stations will fail to meet. A large CW results in low overall efficiency of the WLAN because most low priority stations have a long listening period even if the network is under non-congested condition. It is understandable that to guarantee the real-time requirements of high priority periodic traffic, the low priority periodic traffic only consumes a little WLAN resources and as a result the network utilization is deteriorated.

Another drawback of the EDCA is its lack of an intra-TC QoS differentiation method, particularly when there are a number of high priority stations with different periodic traffic flows in the WLAN. Consequently, the real-time performance of the WLAN is deteriorated because of this drawback. This is because the retry limit L for the EDCA is pre-defined. If a high priority traffic with a long period is assigned with a small L, the packets may be dropped even if the deadline is not reached. On the contrary, a large L results in a high collision rate of the WLAN.

The design goal of the proposed MAC scheme, which we refer to as QoS-dif, is to achieve QoS differentiation to fulfil different deadline requirements of the periodic real-time traffic flows in a single TC. It is expected that with this QoS-dif MAC scheme the real-time performance of the periodic traffic can be guaranteed when the network is under medium (even heavy) traffic load conditions.

The QoS-dif MAC scheme is designed based on the IEEE 802.11e EDCA. It also defines a new backoff algorithm to replace the BEB backoff algorithm in the IEEE 802.11 standard. Furthermore, periodic traffic flows are assigned with different retry limits as a basic MAC parameter in terms of their deadline requirements.

In order to reduce the contention, this paper proposes a contention-sensitive backoff algorithm based on a modification to the BEB backoff algorithm in the IEEE 802.11 standard. In the BEB algorithm, If the medium is sensed busy, a node pauses its backoff timer, and the backoff timer resumes decreasing once the medium is sensed idle. In the proposed backoff algorithm, a node has another transmission attempt instead of the backoff timer pause in this situation. Specifically, if the medium is sensed busy, the node doubles its contention window (CW) and a backoff value is randomly chosen from a uniform distribution in the interval [0, CW]. The node commences decreasing its backoff timer with a doubled CW as soon as the medium becomes idle. The retransmission attempts will continue until the retry limit is reached. The contention window size is doubled in case of not only collisions but also the channel being busy. Compared to the BEB backoff algorithm, the proposed backoff algorithm is more sensitive to contention. When the network is under high contention, the proposed backoff algorithm achieves the objective of increasing the contention window with which the low priority stations contend for channel access. Thus, the probability to

get the media access for low priority stations significantly decreases, and better real-time performance of high priority stations is achieved.

In our new backoff algorithm, a node doubles its contention window if the medium is sensed busy. It is noted that the maximum number of retry is the retry limit L, and the maximum number of backoff slots that a node experiences at the jth retry is $CW[i]$. Consequentially, the maximum backoff delay of the periodic traffic can be estimated through

$$T_{delay\_max} = \sum_{j=0}^{L} [CW[i]_j T_{slot} + T_{AIFS[i]}] + LT_{ACK\_to} + T_s, \quad (1)$$

where i denotes the traffic class AC ($i = 0, 1, 2, 3$). For the ith AC, $CW[i]_j$ is the maximum backoff window size at the backoff stage j. L is the retry limit. T_{slot} is the backoff slot time that is defined in the IEEE 802.11 specification [7]. In this work, the value of L is calculated in terms of the deadline requirements of the periodic traffic. Therefore, each of the periodic traffic flows with different deadlines has a different value of L. T_{ACK_to} is the duration of the ACK timeouts. T_s is the maximum value of the time that the channel is sensed busy because of a successful transmission:

$$T_s = T_H + T_{E(LF)} + T_{SIFS} + T_{ACK} + T_{AIFS[3]}, \quad (2)$$

where $T_{E(LF)}$ is the time duration to transmit a periodic packet; $T_{AIFS[3]}$ is the maximum value among the AIFS times of all traffic classes; T_{SIFS} is SIFS times; T_{ACK} is the time duration to transmit an ACK; and T_H is the time duration to transmit the packet header.

With the proposed backoff algorithm, periodic traffic flows are assigned with retry limits that are estimated in terms of their deadline requirements. $T_{deadline}$ is the deadline for the periodic traffic. A simple algorithm is used to estimate the retry limit for the periodic traffic, as shown in Algorithm 1.

Algorithm 1: Retry limit estimation

```
Result: Retry limit L
L = 1;    //Initialization
while do
    Compute the T_delay_max by using equation (1);
    if T_delay_max ≤ T_deadline then
        ++L;    //Update the value of L
    else
        −−L;    //The maximum value of L is achieved
        with a bounded delay marginally guaranteed;
        break;
    end
end
```

Corresponding to the four TCs in the IEEE 802.11e EDCA, four traffic classes have been introduced to describe different types of communications usually found in real-time control applications: urgent asynchronous notifications (AC_VO), periodic data (AC_VI), sporadic data(AC_BE), and parametrization services (AC_BK) [6]. Therefore, in this paper, all periodic real-time traffic flows belong to a single TC (AC_VI) with high priority. The retry limits for the periodic traffic are computed as described above while the retry limits for other three types of traffic flows are set to 7, as usually done in other wireless network applications. Apart from the retry limit, other IEEE 802.11e EDCA parameters have been shown previously in Table I.

IV. Performance Evaluation

In this paper, the performance of the proposed QoS-dif MAC scheme is evaluated in ideal channel environments without transmission errors or hidden terminals. The parameters of the 802.11b standard are used with the data rate of 11 Mbps and the basic rate of 11 Mbps for demonstrations. T_{SIFS} and T_{slot} (slot time) are set to be 10 μs and 20 μs, respectively. T_{ACK_to} is set to be 300 μs. Simulations are carried out by using Network Simulator Version 2 (NS2), a popular network simulation tool. The simulated network topology is set as follows: the access point is placed at the center of a 100m $\times$ 100m area; and all stations that generate traffic flows are randomly placed on a circle with the radius of 50m from the access point.

Without loss of generality, only two traffic classes AC_VI and AC_BE are considered in the simulations for performance evaluation. They have the same packet size of 200 bytes. Each station has only one type of traffic flows. The number of AC_VI and AC_BE stations are set to be 20 and 10, respectively, in the simulations.

A. Scenario 1: traffic flows each with a different period

Scenario 1 aims to compare the EDCA and the proposed QoS-dif MAC scheme in terms of the real-time performance of periodic traffic flows under different transmission periods. For this purpose, only one type of wireless station class, AC_VI, is investigated. the transmission periods of the periodic traffic flows are 20, 16, 12, 10, 9.5, 9, 8.5, 8ms, respectively. The deadlines for the periodic traffic flows are the same as the corresponding transmission periods. The retry limits for the periodic traffic are set to be 21, 17, 12, 10, 10, and 9, which are estimated from Algorithm 1. Following a Poisson distribution, the input traffic from all the 10 AC_BE class stations is 547.13 kbps. Network dynamics under different traffic load conditions can be observed from the simulations.

Table II shows the average delay and packet loss ratio of the periodic traffic under different transmission periods for both the EDCA and the QoS-dif MAC scheme. It is seen from the table that in terms of average delay and packet loss ratio, both the EDCA and the QoS-dif MAC scheme behave with comparable real-time performance under light traffic conditions. However, when the network is under more congested conditions, the measured average delay T_{avg_delay} and packet loss ratio R_{loss} for the EDCA are dramatically increasing, and are much higher than those for the QoS-dif MAC scheme. For example, when T_{period} = 9ms, T_{avg_delay} is 3.076ms for the EDCA, representing over three times of the average delay 0.997ms for the QoS-dif MAC scheme. Also,

in comparison with the 0.84% packet loss ratio from the QoS-dif MAC scheme, 66.45% periodic packets are dropped for the EDCA. Therefore, the QoS-dif MAC scheme significantly outperforms the EDCA under heavy traffic conditions.

TABLE II
PERFORMANCE COMPARISONS BETWEEN THE EDCA AND THE QOS-DIF MAC SCHEME (T_{period}: TRANSMISSION PERIOD OF THE PERIODIC TRAFFIC FLOWS; T_{avg_delay}: AVERAGE DELAY; R_{loss}: PACKET LOSS RATIO)

T_{period} (ms)	Network Utilization (%)	T_{avg_delay} (ms)		R_{loss} (%)	
		EDCA	QoS-dif	EDCA	QoS-dif
20	27.66	0.362	0.399	0.00	0.00
16	33.34	0.399	0.464	0.00	0.00
12	42.78	0.552	0.651	0.00	0.05
10	50.35	1.259	0.845	8.08	0.18
9.5	51.28	2.478	0.917	50.05	0.46
9	55.40	3.076	0.997	66.45	0.84
8.5	58.36	3.099	1.105	73.34	1.86
8	61.70	2.922	1.237	77.60	3.90

B. Scenario 2: two groups of traffic

Scenario 2 investigates a network with two types of wireless stations C1 and C2, which belong to the same traffic class (AC_VI) but have different transmission periods. There are 10 C1 stations and the same numbers of C2 stations. The periods of the periodic traffic for C2 stations ($T_{period}(C2)$) are fixed at 7ms while the periods for C1 stations ($T_{period}(C1)$) change from 20ms to 15ms. As in Scenario 1, the 10 AC_BE class stations also generate a 547.13 kbps traffic load.

The simulation results for Scenario 2 are depicted in Figure 1. With the decrease of $T_{period}(C2)$, the traffic load in the network increases. As shown in Figure 1, this leads to real-time performance degradation in both the EDCA and the QoS-dif MAC scheme. Under light traffic conditions, both EDCA and the QoS-dif MAC scheme give comparable average delay and packet loss ratio performance. However, under heavy traffic conditions, the QoS-dif MAC scheme is significantly superior to the EDCA. For example, for $T_{period}(C2) = 15$ms, the delay performance of the QoS-dif is much better than that of the EDCA for both C1 and C2 stations. Under the same condition ($T_{period}(C2) = 15$ms), the QoS-dif gives the packet loss ratio of 0.35% for C1 stations in comparison with 14.61% for the EDCA, and 0.43% for C2 stations in comparison with 6.66% from the EDCA, respectively.

V. CONCLUSION

In this paper, a QoS-dif MAC scheme for IEEE 802.11 based WLANs has been proposed to support QoS differentiation among the periodic traffic flows in a single TC in real-time control applications. The real-time QoS for the periodic traffic is effectively improved in real-time control communication scenarios, particularly when the network is under congested conditions. To make the best use of limited wireless resources, a real-time control system tends to operate at a critical traffic condition at its maximum capacity under real-time constraints [3]. Theoretical investigations into the critical real-time traffic condition for the proposed QoS-dif MAC scheme in wireless networked control systems are in progress.

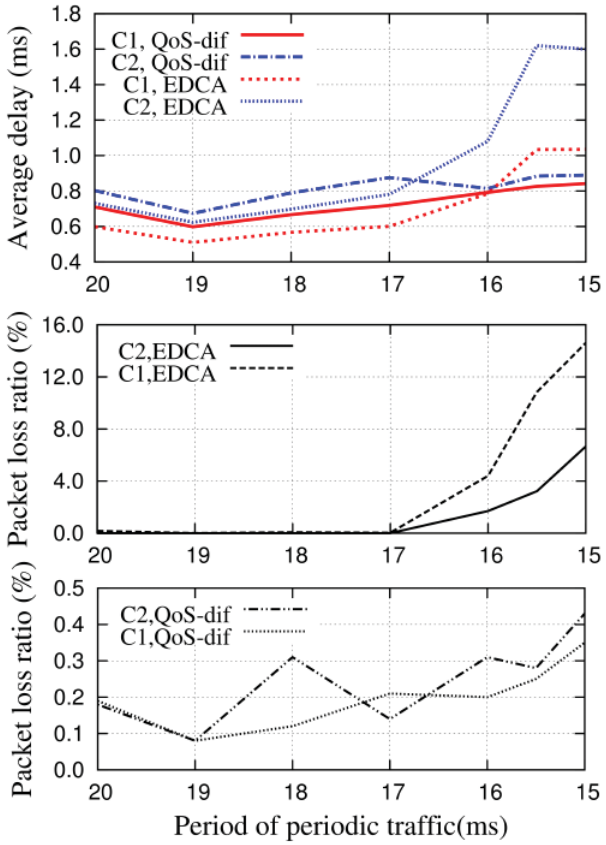


Fig. 1. Comparisons between the EDCA and the QoS-dif in Scenario 2.

REFERENCES

[1] G. Cena, L. Seno, A. Valenzano, and C. Zunino, "On the performance of IEEE 802.11e wireless infrastructures for soft-real-time industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 3, pp. 425–437, August 2010.

[2] G. Tian, Y.-C. Tian, and C. Fidge, "Performance analysis of IEEE 802.11 DCF based WNCS networks," in *Proc. of the The 35th IEEE Conf. on Local Computer Networks (LCN'10)*, Denver, Colorado, U.S.A, October 11-14 2010, pp. 512–519.

[3] G. Tian and Y.-C. Tian, "Modelling and performance evaluation of the IEEE 802.11 DCF for real-time control," *Computer Networks*, vol. 56, no. 1, pp. 435–447, Jan 2012.

[4] R. Moraes, P. Portugal, F. Vasques, and J. A. Fonseca, "Limitations of the IEEE 802.11e EDCA protocol when supporting real-time communication," in *Proc of the IEEE Int Workshop on Factory Commun Sys WFCS' 08*, Dresden, Germany, May 20–23 2008, pp. 119–128.

[5] M. Jonsson and K. Kunert, "Meeting reliability and real-time demands in wireless industrial communication," in *Proc of the 13th IEEE Int Conf on Emerging Tech and Factory Automation ETFA'08*, Hamburg, Germany, Sept 15–18 2008, pp. 877–884.

[6] G. Cena, I. Cibrario, Bertolotti, A. Valenzano, and C. Zunino, "Industrial applications of IEEE 802.11e WLANs," in *IEEE Int Workshop on Factory Commun Sys WFCS'08*, Dresden, Germany, May 20-23 2008, pp. 129–138.

[7] "802.11e-2005 - IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements," IEEE, Nov 2005.

Scheduling of Elastic Mixed-Criticality Tasks in Multiprocessor Real-Time Systems

Hang Su and Dakai Zhu
University of Texas at San Antonio
{hsu, dzhu}@cs.utsa.edu

***Abstract*—To address the service abrupt problem for low-criticality tasks in existing mixed-criticality scheduling algorithms, we have studied an *Elastic Mixed-Criticality (E-MC)* task model, where a low-criticality task may have variable periods and its minimum service requirement is ensured by its largest period [11]. In this paper, based on various partitioning heuristics, we study partition-based schemes for scheduling such tasks in multiprocessor systems. In addition to allowing low-criticality tasks to release early on its host processors (i.e., local early-release, L-ER), we consider also the scheme that allow such tasks to reclaim slack and thus temporarily execute early on other processors (i.e., foreign early-release, F-ER). Our preliminary results show that, compared to *First-Fit (FF)*, the *Worst-Fit (WF)* based heuristics perform worse in acceptance ratio of schedulability. However, better frequency improvements can generally be obtained for low-criticality tasks under WF heuristics, especially when F-ER is enabled.**

I. Introduction

As the next-generation engineering systems, cyber-physical systems (CPS) can have computation tasks with different levels of importance according to their functionalities that further lead to different criticality levels [1]. To incorporate various certification requirements and enable efficient scheduling of such tasks, the *mixed-criticality* task model has been studied recently [3], [7], [10], where a task generally has multiple worst case execution times (WCETs) according to different certification levels.

Since tasks with mutiple criticality levels need to share computing resources, how to efficiently schedule such mixed-criticality tasks while satisfying their specific requirements has been identified as one of the most fundamental issues in CPS [3]. Note that, without proper provisions for such mixed-criticality tasks, traditional scheduling algorithms are likely to cause the so-called *"priority inversion"* problems [7].

As the first work to address the scheduling of such tasks, Vestal formalized the mixed-criticality scheduling problem with mutiple certification requirements at different degrees of confidence and studied a fixed priority scheduling algorithm in [12]. More recently, Baruah *et al.* proposed a more efficient scheduling algorithm, namely EDF-VD (virtual deadline), that assigns *virtual* (and smaller) deadlines for high-criticality tasks to ensure their schedulability in the worst case scenario [2]. De Niz *et al.* proposed a zero-slack scheduling approach for fixed-priority based preemptive algorithm (such as RMS) [7].

As multicore processors become popular for modern computing systems, several works have been studied on the scheduling problem of mixed-criticality tasks in multiprocessor systems. In [5], Baruah *et al.* studied a global-based scheme with Own Criticality Based Priority (OCBP) for a finite collection of independent jobs and partitioned-based EDF-VD for sporadic tasks. Then, Li *et al.* proposed Global EDF-VD as the extension of Global-EDF for scheduling mixed-criticality tasks on multiprocessor systems based on schedulability condition of EDF-VD [9]. Focusing on fixed-priority scheduling, Kelly *et al.* studied the performance of various partitioning heuristics (e.g., First-Fit, Best-Fit and Worst-Fit) and task sorting policies (e.g., Decreasing-Utilization and Decreasing-Criticality) [8].

Note that, in most existing mixed-criticality scheduling algorithms, when any high criticality task uses more time than its low-level WCET and causes the system to enter high-level execution mode, *all* low-criticality tasks will be discarded to provide the required computation capacity for high-criticality tasks [2], [4], [9]. Such an approach can cause serious service abrupt and significant performance loss for low-criticality tasks, especially for control systems where the performance of controllers is mainly affected by the execution frequency and period of control tasks [13].

Based on the traditional mixed-critical task model, Santy *et al.* studied an online scheme that calculates a delay-allowance before entering the high-level execution mode and thus delays the cancellation of low-criticality tasks to improve their services [10]. In our recent work [11], we proposed an *Elastic Mixed-Critical (E-MC)* task model and studied the *early-release EDF (ER-EDF)* scheduling algorithm. The central idea of E-MC is to have variable periods (i.e., service intervals) for low-criticality tasks, where their minimal service levels are represented by their maximum periods and guaranteed offline. At runtime, by properly reclaiming the slack, ER-EDF allows low-criticality tasks to release earlier than their maximum periods and thus to improve their service levels.

In this work, focusing on partitioned scheduling, we studied the performance of various partitioning heuristics for a set of E-MC tasks running on multiprocessor systems. In addition, when there is not enough slack on a low-criticality task's host processor, we further investigate the scheme that allows it to temporarily migrate and reclaim slack on other processors. Our preliminary results show that such online migration scheme can significantly improve the execution frequencies (i.e., service levels) of low-criticality tasks.

II. System and Task Models

We consider a system with m identical processors and a set of *Elastic Mixed-Criticality (E-MC)* tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$.

This work was supported in part by NSF awards CNS-0855247, CNS-1016974 and NSF CAREER Award CNS-0953005.

Each E-MC task τ_i is represented by a 5-tuple parameters: $\tau_i = (\zeta_i, C_i(LO), C_i(HI), VP_i, k_i)$ [11], where ζ_i denotes the criticality level of τ_i. In this work, the same as in [11], we consider systems with two different criticality levels, which are represented by HI and LO, respectively.

For a high-criticality task τ_i (i.e., $\zeta_i = HI$), its $C_i(LO)$ and $C_i(HI)$ specify the worst case execution times (WCET) that are provided by system designers and certification authorities, respectively. We assume that $C_i(LO) \leq C_i(HI)$, which means certification authorities are more pessimistic and strictive than system designers. We further assume that $k_i = 1$ for τ_i, which means that there is only one period in its vector of periods VP_i. The LO and HI utilizations of τ_i are defined as $u_i^{LO} = C_i(LO)/VP_i[k_i]$ and $u_i^{HI} = C_i(HI)/VP_i[k_i]$, respectively.

For a low-criticality task τ_i (i.e., $\zeta_i = LO$), its WCETs are assumed to be $C_i(LO) = C_i(HI)$. There are $k_i(\geq 1)$ periods for τ_i, which are represented as a vector $VP_i = \{VP_i[1], \cdots, VP_i[k_i]\}$. Here, we assume that $C_i(LO) \leq VP_i[1] <, \cdots, < VP_i[k_i]$. That is, $VP_i[k_i]$ denotes τ_i's maximum period that ensures the minimum service level required by the task. Other periods represent $(k_i - 1)$ possible *early-release (ER) points* for task τ_i provided that there is enough reclaimable slack at runtime. The minimum utilization of τ_i is defined as $u_i^{MIN} = C_i(LO)/VP_i[k_i]$.

Moreover, for any task τ_i (either high-criticality or low-criticality), we assume that any instance of τ_i will not run longer than $C_i(HI)$ (otherwise, such instance will be aborted and the error will be reported). In addition, when a task instance of τ_i releases at time t, its deadline is assumed to be $t + VP_i[k_i]$.

Suppose that a given task-to-processor partition is $\Pi = \{\Gamma_1, \cdots, \Gamma_m\}$, where $\Gamma = \Gamma_1 \cup, \cdots, \cup \Gamma_m$. From [11], we have the following theorem:

Theorem 1: For a set Γ of E-MC tasks running on m processors, a given task-to-processor partition Π is feasible under partitioned-EDF, if $\forall k$ $(k \in [1, \cdots, m])$, $U_{\Gamma_k}(H, H) + U_{\Gamma_k}(L, MIN) \leq 1$, where $U_{\Gamma_k}(H, H) = \sum_{\tau_i \in \Gamma_k \wedge \zeta_i = HI} u_i^{HI}$ and $U_{\Gamma_k}(L, MIN) = \sum_{\tau_i \in \Gamma_k \wedge \zeta_i = LO} u_i^{MIN}$.

A. Early-Release of A Low-Criticality Task

Note that, high-criticality tasks may only use $C_i(LO)$ for most of the time and excessive slack can be expected at runtime. Therefore, such slack can be exploited by low-criticality tasks to release their instance more frequently (and thus improve their service levels) [11]. Intuitively, releasing a task instance at an earlier time point introduces extra workload to the system. Therefore, to avoid overload condition and affecting the execution of other (especially high-criticality) tasks, such early-release decisions require judicious slack allocation.

Suppose that task τ_i has been partitioned to processor P_x, where its current task instance released at time r_i and completed the execution at time t $(r_i < t \leq r_i + VP_i[k])$. Note that, to ensure its minimum service level, τ_i's current task instance has the deadline at time $r_i + VP_i[k_i]$, which supposes to be the release time of its next task instance. That is, the default service interval is its maximum period $VP_i[k_i]$ [11].

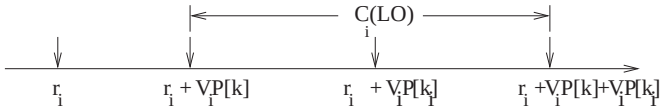


Fig. 1: The deadline of an early-release task instance [11].

As shown in Figure 1, if its next task instance is released at the early-release point $r_i + VP_i[k]$, the new deadline will be $r_i + VP_i[k] + VP_i[k_i]$. Therefore, as shown in the figure, τ_i's new task instance needs $C_i(LO)$ from time $r_i + VP_i[k]$ to $r_i + VP_i[k] + VP_i[k_i]$. From [6], we know that the processor share that is guaranteed for task τ_i during such time interval can be found as $VP_i[k] \cdot u_i^{MIN}$. Therefore, the amount of slack needed before the expected deadline $r_i + VP_i[k] + VP_i[k_i]$ is $C_i(LO) - VP_i[k] \cdot u_i^{MIN}$. As long as the amount of reclaimable slack for task τ_i is no less than $C_i(LO) - VP_i[k] \cdot u_i^{MIN}$, it can safely release its next task instance at time $r_i + VP_i[k]$ [11].

III. Partition-Based Scheduling of E-MC Tasks

A. Partitioning Heuristics

The task-to-processor partitioning heuristics considered in this work are similar to those in [8]. That is, when assigning a given task to processors, we consider *First-Fit (FF)*, *Best-Fit (BF)* and *Worst-Fit (WF)* heuristics. Moreover, for the order of tasks being allocated, the following heuristics are considered:

- Decreasing Utilization (DU): all tasks are sorted in non-increasing order of their utilizations; here, we use u_i^{HI} and u_i^{MIN} for high- and low-criticality tasks, respectively;
- Decreasing Criticality (DC): high-criticality tasks are allocated before low-criticality tasks; for tasks with the same criticality level, they are sorted in non-increasing order of their utilizations;
- Decreasing Difference (DF): high-criticality tasks are allocated before low-criticality tasks; for high-criticality tasks, they are sorted in non-increasing order of the difference between C_i^{HI} and C_i^{LO}; low-criticality tasks are sorted in non-increasing order of their utilizations.

Note that, with three task ordering and three task allocation heuristics, there are total ten different task-to-processor partitioning schemes. The performance (such as task set acceptance ratio and execution frequency improvements of low-criticality tasks) under these schemes are evaluated in Section IV.

B. Early-Release: Local vs. Global

In this work, for a given feasible task-to-processor partition under partitioned-EDF scheduling, we assume that the wrapper-task based slack management technique [14] is utilized on every processor. Therefore, the straightforward approach is to deploy ER-EDF on every processor *independently* [11], where a low-criticality task only checks the available slack on its *host* processor for early-release opportunities.

Algorithm 1 : Partitioned-EDF with Global Early-Release

1: **Input:** τ_i, P_x and $t = r_i + VP_i[k]$;
2: $S_{need}^{local} = C_i(LO) - VP_i[k] \cdot \frac{C_i(LO)}{VP_i[k_i]}$;
3: $S_{need}^{foreign} = C_i(LO)$; $d_i^{exp} = t + VP_i[k_i]$;
4: **if** ($S_{need}^{local} \leq$ *CheckSlack(*$SQ_x(t)$, d_i^{exp}*)*) **then**
5: *ReclaimSlack(*$SQ_x(t)$, S_{need}^{local}*)*;//local early-release
6: *Enqueue(*$ReadyQ_x$, τ_i*)*; //add τ_i to P_x's ready queue
7: **else if** ($\exists y, S_{need}^{foreign} \leq$ *CheckSlack(*$SQ_y(t)$, d_i^{exp}*)* $\wedge y \neq x$) **then**
8: *ReclaimSlack(*$SQ_y(t)$, $S_{need}^{foreign}$*)*;//foreign early-release
9: *Enqueue(*$ReadyQ_y$, τ_i*)*; //add τ_i to P_y's ready queue
10: **else**
11: *SetTimer(*$r_i + VP_i[k+1]$*)*;//set next early-release time
12: **end if**

Note that, the amount of available slack can be different on the processors depending on the heuristics when partitioning tasks to processors. Therefore, it is possible that a low-criticality task cannot obtain enough slack for early-release on its host processor but there is enough slack on other (foreign) processors. For such cases, if task migration is allowed, we can *temporarily* migrate the low-criticality task to a foreign processor and reclaim the available slack to release the next task instance early. Once the migrated task instance completes, it will migrate back to its host processor. The major steps for the partitioned-EDF with *Global Early-Release* are summarized in Algorithm 1, which will be invoked at any early-release point $t = r_i + VP_i[k]$ ($1 \leq k < k_i$) of a low-criticality task τ_i on its host processor P_x.

Here, the amount of slack needed for task τ_i to *safely* release its next task instance at the early-release time point t ($= r_i + VP_i[k]$) on its host and other (foreign) processors, which are denoted as S_{need}^{local} ($S_{need}^{foreign}$), respectively, are first calculated (lines 2 and 3). Moreover, the expected deadline d_i^{exp} of its next task instance is also calculated. Note that, if τ_i releases its next instance locally on its host processor P_x, the amount of needed slack is the same as in ER-EDF [11]. However, if τ_i intends to borrow slack from other (foreign) processors, the amount of $C_i(LO)$ is needed as there is no reserved processor share for τ_i on other processors.

Task τ_i first considers its host processor P_x and tries to release its next instance locally. That is, if there is enough reclaimable slack before d_i^{exp} on P_x, τ_i will reclaim such slack and release its next task instance on P_x (lines 5 and 6). Here, $SQ_x(t)$ represented the available slack queue on processor P_x. *CheckSlack()* and *ReclaimSlack()* are functions to check and reclaim the amount of available slack before a given time, respectively [14]. *Enqueue()* is the common function to insert a ready task.

Otherwise, τ_i will check the amount of available slack on other processors one by one. If it can find a processor that has enough available slack, it will release its next instance on the foreign processor (lines 8 and 9); If there is no processor that has enough slack, a timer will be set for τ_i's next early-release time point (line 11).

From [11], we know that allowing τ_i to release its next task instance early on its host processor P_x will not cause any deadline miss on P_x. Moreover, following the similar reasonings as in [6], [11], we can find that allowing τ_i to temporarily migrate to another processor P_y will not lead to deadline misses on P_y as well since such migration does not introduce additional workload for P_y. Therefore, there is no deadline miss for any (especially high-criticality) task under Algorithm 1.

IV. Evaluations and Discussions

In this section, we present some preliminary simulation results, which show the performance of various partitioning heuristics and the effectiveness of the proposed Global Early-Release technique. For comparison, we also implemented the Global EDF-VD (denoted as *GLO*) [9], the state-of-the-art mixed-criticality multiprocessor scheduler. Here, whenever a high-criticality task τ_i executes more than its $C_i(LO)$ and causes the system to enter HI execution mode at run-time, GLO will discard all (current and future coming) low-criticality tasks until there is no more *ready* high-criticality tasks (at that time, the system safely switches back to LO mode) [9]. Task sets are randomly generated following the similar method as in [9]. The u_i^{HI} of a task τ_i (either $\zeta_i \in$ LO or HI) is uniformly generated in $[U_{lower}, U_{upper})$. Then, u_i^{LO} is obtained by $\frac{u_i^{HI}}{Z_i}$ for task τ_i ($\zeta_i = HI$), where Z_i uniformly generated in $[Z_{lower}, Z_{upper})$. The normalized system utilization is U_{bound}/m. The $prob(HI)$ denotes the probability of a task being a high-criticality task.

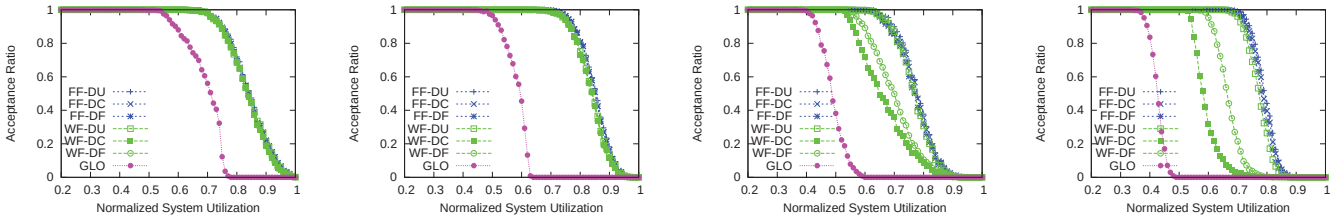
A. Acceptance Ratio

First, Figure 2 shows the acceptance ratio of schedulable task sets under partitioned-EDF with various partitioning heuristics (see Theorem 1) and GLO [9]. The U_{lower} and U_{upper} are fixed as 0.05 and 0.5, respectively. Moreover, $Z_{lower} = 1$ and $Z_{upper} = 8$. Since the heuristic variations with BF have very close acceptance ratio as those with FF, we only show the results for FF-variations.

We can see that the acceptance ratios of E-MC tasks under partitioned-EDF are always better than that of Global-EDF-VD. Moreover, when the number of high-criticality tasks is small, there is no big difference between different partitioning heuristics. However, when there are more high-criticality tasks, the acceptance ratios under FF-variations become better than WF-variations. Moreover, the DU task ordering performs the best for WF since it is known to generate partitions with balanced workload among processors. In comparison, the DC task ordering performs the worst since its first priority is to balance high- and low-criticality tasks among processors.

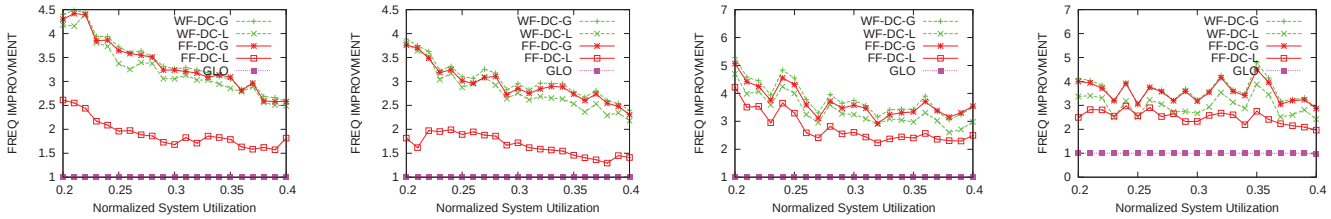
B. Frequency Improvments for Low-Criticality Tasks

In this section, we evaluate the execution frequency improvements (i.e., how many additional instances are executed) for low-criticality tasks. We used the result under no early-release as baseline.



a. m=2, $prob(HI)$ = 0.2; b. m=4, $prob(HI)$ = 0.2; c. m=4, $prob(HI)$ = 0.8; d. m=16, $prob(HI)$ = 0.8;

Fig. 2: Acceptance ratio under different scheduling and partitioning schemes.



a. m=2, $prob(HI)$ = 0.2; b. m=4, $prob(HI)$ = 0.2; c. m=2, $prob(HI)$ = 0.8; d. m=4, $prob(HI)$ = 0.8;

Fig. 3: Normalized frequency improvement for low-criticality tasks under different schemes.

The periods of tasks are uniformly generated in [20, 100). For high-criticality tasks, 50% of their instances take $C_i(LO)$. For low-criticality tasks, we set $k_i = 20$ (i.e., 19 early-release points that are evenly distributed between $VP_i[1]$ and $VP_i[20]$. For Partitioned-EDF, the ones with independent local early-release are denoted as *-*L*, while the ones with Global Early-Release enabled are denoted as *-*G*. Here, we consider normalized system utilization up to 0.4 to obtain scheduleable task sets for all schemes and the results are shown in Figure 3.

First, without incorporating early-release technique, GLO does not obtain any frequency improvement for low-criticality tasks. In fact, when there are more high-criticality tasks (see Figure 3d) at high system utilization, low-criticality tasks can even be cancelled and lead to worse performance (i.e., less then 1.0).

Second, for the partition-based schemes, the Global Early-Release technique can obtain better frequency improvements for low-criticality tasks, especially for FF-DC. The reason comes from its ability to share slack among processors. However, such benefits are limited for WF-DC since the high-criticality tasks (thus the potential online slack) are already evenly distributed among processors.

V. Conclusions

In this work, based on the *Elastic Mixed-Criticality (E-MC)* task model, we study *partition-based Early-Release EDF* scheduing algorithms on multiprocessor sysetms. We consider various partitioning heuristics and propose a Global Early-Release technique, which allows low-criticality tasks to borrow slack from other (foreign) processors and temporarily migrate its execution for better execution frequency improvements. Our preliminary simulation results show the effectiveness of G-ER in scheduling mixed-criticality tasks when comparing to the state-of-the-art Global EDF-VD algorithm [9].

References

[1] J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. Scoredos, P. Stanfill, D. Stuart, and R. Urzi. A research agenda for mixed-criticality systems. *Cyber-Physical Systems Week*, 2009.

[2] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. M.-Spaccamela, S. van der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *ECRTS*, pages 145–154, 2012.

[3] S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *RTAS*, pages 13–22, 2010.

[4] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *ECRTS*, pages 147–155, 2008.

[5] S.K. Baruah, V. Bonifaci, G. DAngelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. In *ESA*, pages 555–566, 2011.

[6] S.A. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time, and non-real-time processes. In *RTSS*, pages 396–407, 2003.

[7] D. de Niz, K. Lakshmanan, and R. Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *RTSS*, pages 291–300, 2009.

[8] O.R. Kelly, H. Aydin, and B. Zhao. On partitioned scheduling of fixed-priority mixed-criticality task sets. In *ICESS*, 2011.

[9] H. Li and S. Baruah. Global mixed-criticality scheduling on multiprocessors. In *ECRTS*, 2012.

[10] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp. In *ECRTS*, pages 155–165, 2012.

[11] H. Su and D. Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In *DATE*, 2013.

[12] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS*, 2007.

[13] F. Zhang, K. Szwaykowska, W. Wolf, and V. Mooney. Task scheduling for control oriented requirements for cyber-physical systems. In *RTSS*, pages 47–56, 2008.

[14] D. Zhu and H. Aydin. Reliability-aware energy management for periodic real-time tasks. *IEEE Trans. on Computers*, 58(10):1382–1397, 2009.

Supporting Development of Energy-Optimised Java Real-Time Systems using TetaSARTS

Kasper Søe Luckow, Thomas Bøgholm, and Bent Thomsen
Department of Computer Science, Aalborg University
{luckow,boegholm,bt}@cs.aau.dk

***Abstract*—This paper presents how the tool TetaSARTS can be used to support the development of embedded hard real-time systems written in Java using the emerging Safety Critical Java (SCJ) profile. TetaSARTS facilitates control-flow sensitive schedulability analysis of a set of real-time tasks, and features a pluggable platform specification allowing analysis of systems including the hosting execution environment. This is achieved by approaching the analysis as a model checking problem by modelling the system using the Timed Automata formalism of the model checking tool UPPAAL. The resulting Timed Automata model facilitates easy adjustment of a wide variety of parameters that may be of interest such as processor frequency.**

This paper demonstrates that TetaSARTS can be used for tuning processor frequency, for conducting control-flow sensitive Worst Case Response Time analysis, and for conducting processor utilisation and idle time analysis.

I. INTRODUCTION

It is well-known, that the traditional Java run-time is unsuited for use in embedded real-time systems, which is attributed issues related to the lack of high-resolution real-time clocks and timers, insufficient thread semantics, and, most notably, memory management, which is traditionally handled by a garbage collector whose execution is highly unpredictable. However, with emerging standards such as the Real-Time Specification for Java (RTSJ) [6] and the Safety Critical Java (SCJ) [12] profile, these issues have been accounted for, thereby achieving a significant step towards use in embedded real-time systems development.

However, having a programming model, as introduced by e.g. SCJ, is not the only component in making Java a viable technology and competitor to C in the embedded real-time systems market. Of equal importance is the complementation of tools and analyses for verifying that the system is correct. For real-time systems, this entails functional correctness, but also temporal correctness for which showing that the system is schedulable, that is, showing that no deadline violations can occur, is of utmost importance.

Other analyses are also important. Since embedded systems are often produced in large quantities, mitigating the unit price is also an imperative. It is hence desirable showing that the system is schedulable on a platform containing the fewest possible resources. Using a similar rationale, it is also desirable to reduce the running costs of the system after deployment. This can partly be achieved by limiting the energy consumption, which in turn is partly a result of running the system with the lowest possible processor clock frequency, while still ensuring that the system is schedulable.

In this paper, we present how TetaSARTS[1] can be used for conducting the presented analyses. The tool facilitates control-flow sensitive schedulability analysis of a set of SCJ real-time tasks analysed on a pluggable platform model that allows taking into account an exact, control-flow sensitive representation of the underlying execution environment. TetaSARTS supports a traditional execution environment configuration composed of a Java Virtual Machine (JVM), such as the Hardware near Virtual Machine (HVM) [13], running on embedded hardware, such as Atmel's AVR range of microcontrollers, and it also accommodates hardware implementations of the JVM, such as the Java Optimized Processor (JOP) [16]. Moreover, it also allows different schedulers to be used, and many parameters related to the execution of the system, such as processor clock frequency of the hardware, are adjustable.

The rest of the paper is organised as follows; in Section II, we present related work, followed by an overview of the TetaSARTS tool in Section III. Afterwards, we present the capabilities of TetaSARTS and the analyses it supports in Section IV. In Section V, we present initial results of using the presented usages on two representative examples of real-time systems. Finally, in Section VI, we conclude on the results and make pointers to future development.

II. RELATED WORK

TetaSARTS is inspired by tools for timing analysis including TetaJ [11], METAMOC [8], SARTS [4], TIMES [1], and UPPAAL [2]. TetaJ is a Worst Case Execution Time (WCET) analysis tool for Java Bytecode systems compiled from SCJ programs and employs a model-based approach for analysis inspired by METAMOC which analyses C programs. In TetaJ, the Java Bytecode system, the JVM implementation, and the hardware are modelled as a Network of Timed Automata (NTA) amenable to model checking using UPPAAL. A Timed Automaton (TA) is a finite state machine extended with real-valued clocks, and an NTA is the parallel composition of n TAs sharing clocks and actions (see [2] for more). SARTS is a schedulability analysis tool employing a model-based approach inspired by the model-based ideas of TIMES. While TIMES relies on a static WCET component and is based on abstract descriptions of the behavior of the system, SARTS simulates a control-flow sensitive execution of the Java Bytecode system. It, however, assumes that the Java Bytecode

[1]TetaSARTS is available at http://people.cs.aau.dk/~luckow/tetasarts/

execution times are fixed, and that the execution environment is based on the JOP.

Harnessing the model used for schedulability analysis for other purposes, is to some extent inspired by the work of [15] and [10]. In that work, schedulability analysis is performed by constructing an NTA which simulates the behavior of the real-time tasks in the system. This simulation also allows for determining the Worst Case Response Time (WCRT) of tasks, and processor utilisation and idle time. The models, however, are not directly generated from program source, and real-time task parameters, such as WCET and the behavior of the real-time tasks, are manually encoded in the NTA.

III. TetaSARTS

TetaSARTS distinguishes itself from existing tools by incorporating a control-flow sensitive notion of the execution environment hosting the real-time system. In its current form, AVR and ARM hardware platforms are supported together with the HVM and JOP JVM implementations. Furthermore, it supports SCJ and real-time tasks with periodic or sporadic release patterns, and also accounts for blocking, as introduced by synchronised methods in Java.

It adopts model checking for schedulability analysis; the Java Bytecode system and the JVM implementation are transformed to an NTA, and combined with an NTA simulating the hardware. This transformation is automatic, thereby ensuring that a tight correspondence is kept between the actual system and the model used for analysis. The transformation process draws similarities with that of an optimising compiler: the Java Bytecode system (or the JVM executable), is initially transformed into an intermediate representation (TIR), which is similar to a Control-Flow Graph. This forms the basis for various TA-independent analyses and transformations such as loop identification analysis. Afterwards, TIR is transformed to an NTA, and analyses and optimisations are applied including *TA Inlining*, *Devirtualisation*, *JVM Specialisation*, and *Edge Aggregation*. These contribute to reductions in the size of the state and the state space. The details of these and their effect are documented in [14], which also contains a formalisation of the translation from Java Bytecode system to the NTA.

The architecture of the resulting NTA resembles the original architecture of a Java Bytecode system as depicted in Figure 1. TetaSARTS offers two representations of the execution environment; if the Java Bytecodes have statically fixed execution times, as is the case on e.g. the JOP, the execution environment can be *inlined* thus reducing TA instantiations and communication overhead. In the other case, where the execution times of the Java Bytecodes are dependent on the state of the JVM and hardware, the execution environment is *explicitly* represented.

The `Scheduler TA` simulates the adopted scheduling policy; currently FPS, EDF, and FIFO policies are supported, but the support is extendible. This TA governs the execution of the periodic and sporadic tasks of the system each of which having a corresponding `Task Controller TA`, that handles the execution of the task e.g. periodically releasing it (potentially after an offset), monitoring whether deadlines have

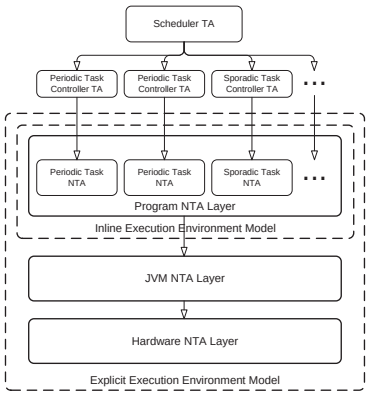


Fig. 1. The architecture of the resulting NTA.

been missed etc. In the `Program NTA`, TAs simulate a control-flow sensitive execution of each of the real-time tasks; for each method, there is a corresponding TA simulating its execution. Listing 1 shows a simple periodic task written in the SCJ profile, and Figure 2 shows an excerpt of the corresponding TA demonstrating how the control-flow structure is captured for the conditional branch.

```
public class MethaneCtrl extends PeriodicEventHandler {
...
    public void handleAsyncEvent() {
        if (this.methaneSensor.isCritMethaneLvlReached())
            this.waterpumpActuator.run();
        else
            this.waterpumpActuator.stop();
    }
}
```

Listing 1. SCJ event handler periodically firing *handleAsyncEvent()*.

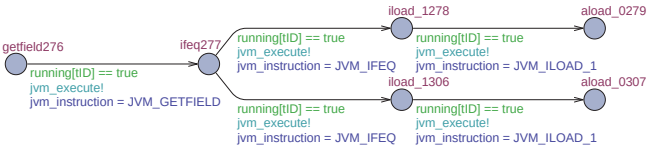


Fig. 2. Excerpt of a TA simulating the execution of Java Bytecodes.

Every firing of an edge in the TA, simulates an abstract execution of the particular instruction. It is abstract in the sense that only the control flow is considered, and the model checker therefore explores all possible execution paths of the tasks. In Figure 2, the execution environment is explicitly represented, and thus the simulation of the Java Bytecode is transferred to the `JVM NTA`, which receives on the *jvm_execute* synchronisation channel, and consults the variable *jvm_instruction* which contains the Java Bytecode to simulate. The structure of the `JVM NTA` is similar to the `Program NTA`; each Java Bytecode simulation is enclosed in a separate TA simulating the execution of the machine instructions by consulting the `Hardware NTA`, which contains models of the pipeline and caching behavior etc. TetaSARTS is capable of automatically constructing the `JVM NTA` provided the JVM executable, and provided that the JVM has a certain structure. The `Hardware NTA`'s are reused from the METAMOC project [8].

IV. USAGE

A. Schedulability Analysis

The primary functionality of TetaSARTS is schedulability analysis of Java Bytecode real-time systems by extracting the real-time requirements of the real-time tasks from the source code, that is, period, offset, deadline, and release pattern. By configuring TetaSARTS in terms of specifying the execution environment constituents and scheduling policy, schedulability analysis is performed by simulating an abstract execution of the Java Bytecode real-time system on the hosting execution environment. The schedulability analysis is expressed using the UPPAAL query specification $\Box$ *not deadlock*, meaning that in all reachable states, the system will not enter a deadlock state. This state can only be reached if a deadline is missed. In Figure 3, the `Task Controller TA` for a periodic task is shown. The deadlock state occurs when the TA enters the

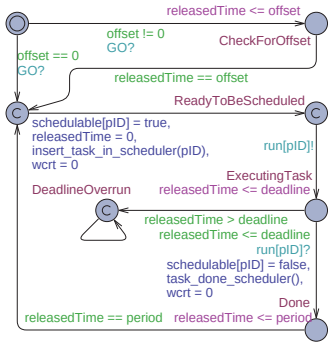


Fig. 3. `Task Controller TA` associated with a periodic task.

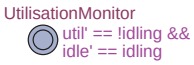


Fig. 4. `Utilisation TA`.

DeadlineOverrun location, which is only the case when the edge with the guard $releasedTime > deadline$ is true.

B. Processor Utilisation Analysis

TetaSARTS can also be used for determining worst case processor utilisation and idle time. When this option is enabled, TetaSARTS generates a new TA, shown in Figure 4, and adds two new clock variables, *idle* and *util*, which are used as stop-watches [9]. When the scheduler has set a real-time task for execution, it also sets the *idling* variable to false, which makes the *idle* clock stop progressing, and makes the *util* progress. The opposite holds when the scheduler is idling, that is, when no task is eligible for execution. Determining the worst case processor utilisation and processor idle time, is hence a matter of determining the maximum values of the newly introduced clocks. This can be done by using the UPPAAL sup-query $sup : util, idle$, which explores the entire state space, and returns the *supremum*, that is, the maximum observed value of the specified variables/clocks.

C. Worst Case Response Time Analysis

Traditional methods for WCRT analysis, such as [7], are usually based on a coarse, control-flow insensitive process model, and do not include detailed information about the underlying hardware, because these are included statically in the WCET component of the analysis. Further, they do not account for the release patterns of sporadic tasks, which are regarded as periodic with period set to the minimum inter-arrival time. This is a safe assumption, but can be a conservative approximation when e.g. the releases of two sporadic tasks, t_1 and t_2, are conditional, such that either t_1 or t_2 is fired, but never at the same time. Usually, such control-flow dependent behavior can not be captured by traditional approaches. TetaSARTS, however, accommodates these shortcomings, and is capable of conducting a control-flow sensitive WCRT analysis that also includes blocking as introduced by the *synchronized* keyword in Java. When this option is enabled, a new clock variable, *wcrt*, is introduced and is reset on the edge going to the *ReadyToBeScheduled* location, and on the edge with destination in the *Done* location (See Figure 3). The WCRT of the task associated with the `Task Controller TA`, can now be determined as the maximum observed value of the *wcrt* clock variable in the *ExecutingTask* location using the sup-query $sup\{periodicThread.ExecutingTask\} : periodicThread.wcrt$. The same applies for sporadic tasks. This value will also account for blocking, and cases where the task is pre-empted as governed by the scheduling policy.

D. CPU Clock Frequency Analysis

For many embedded systems, such as AVR and JOP, the CPU clock frequency is variable, thus, in case energy reductions are imperative, it is desirable to reduce this to a minimum, while still guaranteeing that the system is schedulable. TetaSARTS currently offers an iterative process for determining the appropriate clock frequency; when generating the NTA, the clock frequency of the system used in the analysis can be specified. Using a method like the bisection method, the system can iteratively be analysed for schedulability by incrementing or decrementing the clock frequency until a desired precision is reached.

V. EVALUATION AND RESULTS

We demonstrate the usages of TetaSARTS using the textbook example of a Minepump [7], [3], [11] and the Real-Time Sorting Machine (RTSM) [4]. Both are representative of real-time systems written in Java. Since the SCJ specification is still a draft, the Minepump and the RTSM have been written in a variant of it. This, however, is only a syntactical matter, and does not affect the validity of the results. A system containing an Intel Core i7-2620M @ 2.70GHz and 8 GB of memory has been used in the evaluation.

To demonstrate that TetaSARTS can be used for determining an appropriate CPU clock frequency, we have analysed the Minepump control system hosted by an execution environment consisting of the HVM running on an AVR ATmega2560, and on the JOP, respectively. The results are shown in Table I.

For demonstrating that TetaSARTS can be used for processor utilisation and processor idle time analysis, we use an inline representation of the JOP execution environment. The results of the analysis are shown in Table II.

Execution Environment	Clock Freq.	Schedulable
HVM + AVR	10 MHz	✓
HVM + AVR	5 MHz	×
JOP	2 MHz	✓
JOP	1 MHz	×

TABLE I
USING TETASARTS WITH VARIOUS EXECUTION ENVIRONMENTS.

System	Clock Freq.	Proc. Util.	Proc. Idle
RTSM	100 MHz	48.5 μs	4.0 ms
RTSM	60 MHz	80.8 μs	4.0 ms
Minepump	100 MHz	25.9 μs	2.0 ms
Minepump	10 MHz	259 μs	11.8 ms

TABLE II
RESULTING PROCESSOR UTILISATION AND PROCESSOR IDLE TIMES.

Common to analysing the RTSM and the Minepump is that model checking times are only a few seconds. As expected, the processor is utilised more as the CPU clock frequency is reduced.

For demonstrating WCRT analysis, we use the RTSM system, an inline representation of the JOP, and the CPU clock frequency is set to 60 MHz. The results are shown in Table III.

RT Task	WCRT	Analysis Time
Periodic Task 1	62.3μs	60s
Periodic Task 2	18.4μs	10s
Sporadic Task 1	4.5μs	10s
Sporadic Task 2	4.5μs	25s

TABLE III
RESULTS OF WCRT ANALYSIS OF THE REAL-TIME TASKS OF THE RTSM.

As shown, conducting WCRT analysis can be done relatively quickly.

VI. CONCLUSION

In this paper, we have presented how the tool TetaSARTS can complement the development process of Java real-time embedded systems development. TetaSARTS is primarily targeted at control-flow sensitive schedulability analysis of Java Bytecode systems while taking into account the execution environment. While this analysis forms one of the cornerstones in guaranteeing temporal correctness of the system, TetaSARTS also allows for analysing other interesting parameters: processor utilisation and idle time, the schedulability of the system when adjusting the CPU clock frequency, and for Worst Case Response Time of the real-time tasks. This paper has demonstrated that TetaSARTS is applicable for these analyses using an evaluation featuring representative examples of real-time systems.

Supporting real-time systems development using TetaSARTS is an ongoing effort, and we envision a variety of extensions and improvements. Firstly, we want to examine the effect of using TetaSARTS on more case studies featuring real-life examples of systems with other, and potentially more complex, execution environments. This will entail the development of more hardware models, and automating the process of constructing the `JVM NTA` from other JVM implementations. Secondly, we want to support other relevant analyses as well e.g. blocking time analysis, and memory related analyses such as stack height analysis, and worst case heap consumption analysis. Finally, we also envision TetaSARTS to be extended with *Schedulability Abstractions*, as defined in [5], where the interface of methods in a Java program is decorated with behavioural descriptions to capture requirements for individual methods. However, this requires an extension of the specification language to allow platform dependent timing constraints to be expressed.

REFERENCES

[1] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times - a tool for schedulability analysis and code generation of real-time systems. In *Formal Modeling and Analysis of Timed Systems*, volume 2791 of *Lecture Notes in Computer Science*. 2004.

[2] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL — a tool suite for automatic verification of real-time systems. Lecture Notes in Computer Science. 1996.

[3] T. Bøgholm, C. Frost, R. R. Hansen, C. S. Jensen, K. S. Luckow, A. P. Ravn, H. Søndergaard, and B. Thomsen. Towards harnessing theories through tool support for hard real-time java programming. *Innov. Syst. Softw. Eng.*, 2013.

[4] T. Bøgholm, H. Kragh-Hansen, P. Olsen, B. Thomsen, and K. G. Larsen. Model-based schedulability analysis of safety critical hard real-time Java programs. In *JTRES '08: Proc. of the 6th int'l workshop on Java tech. for real-time and embedded systems*, JTRES '08, 2008.

[5] T. Bøgholm, B. Thomsen, K. Larsen, and A. Mycroft. Schedulability analysis abstractions for safety critical java. In *Proc. of ISORC'12*, 2012.

[6] G. Bollella and J. Gosling. The real-time specification for java. *Computer*, 33(6):47 –54, jun 2000.

[7] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages: ADA 95, Real-Time Java, and Real-Time POSIX*. Addison-Wesley Educational Publishers Inc., 4th edition, 2009.

[8] A. E. Dalsgaard, M. C. Olesen, M. Toft, R. R. Hansen, and K. G. Larsen. METAMOC: Modular Execution Time Analysis using Model Checking. In *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, 2010.

[9] A. David, J. Illum, K. Larsen, and A. Skou. *Model-Based Framework for Schedulability Analysis Using UPPAAL 4.1*. 2009.

[10] A. David, K. Larsen, A. Legay, and M. Mikučionis. Schedulability of herschel-planck revisited using statistical model checking. In *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies*. 2012.

[11] C. Frost, C. S. Jensen, K. S. Luckow, and B. Thomsen. Wcet analysis of java bytecode featuring common execution environments. In *Proceedings of the 9th International Workshop on Java Technologies for Real-Time and Embedded Systems*, JTRES '11, 2011.

[12] JSR302. The java community process™program - jsrs: Java specification requests - detail jsr# 302, 2011. http://www.jcp.org/en/jsr/detail?id=302.

[13] S. Korsholm. Hvm lean java for small devices, 2011. www.icelab.dk.

[14] K. S. Luckow, T. Bøgholm, B. Thomsen, and K. G. Larsen. Flexible schedulability analysis of java bytecode real-time systems. 2013. Submitted to SAS 2013.

[15] M. Mikučionis, K. G. Larsen, J. I. Rasmussen, B. Nielsen, A. Skou, S. U. Palm, J. S. Pedersen, and P. Hougaard. Schedulability analysis using uppaal: Herschel-planck case study. In *Proc. of the 4th int'l conf. on Leveraging applications of formal methods, verification, and validation*, ISoLA'10, 2010.

[16] M. Schoeberl. JOP: A Java Optimized Processor. In *On the Move to Meaningful Internet Systems 2003: Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2003)*, LNCS, Catania, Italy, 2003.

Toward an Optimal Fixed-Priority Algorithm for Energy-Harvesting Real-Time Systems

Yasmina Abdeddaïm
Université Paris-Est,
LIGM UMR CNRS 8049,
ESIEE Paris,
2 bld Blaise Pascal, BP 99,
93162 Noisy-le-Grand Cedex, France
y.abdeddaim@esiee.fr

Younès Chandarli
Université Paris-Est,
LIGM UMR CNRS 8049,
Université Paris-Est Marne-La-Vallée,
5 bld Descartes,
77454 Marne-la-Vallée Cedex 2, France
younes.chandarli@univ-paris-est.fr

Damien Masson
Université Paris-Est,
LIGM UMR CNRS 8049,
ESIEE Paris,
2 bld Blaise Pascal, BP 99,
93162 Noisy-le-Grand Cedex, France
d.masson@esiee.fr

***Abstract*—This paper addresses the real-time fixed-priority scheduling problem for battery-powered embedded systems whose energy storage unit is replenished by an environmental energy source. In this context, a scheduling policy should account for incoming energy, capacity of the battery and tasks cost of energy. Thus, classical fixed-priority schedulers are no longer suitable for this model. Some algorithms were proposed in precedent works to solve this problem, however, none of them has been proved optimal for concrete tasksets[1]. Based on this motivation, we propose FPC_{ASAP} a scheduling algorithm that handles both energy and timing constraints by predicting future possible failure. Such kind of algorithm is said to be clairvoyant. We expect this algorithm to be optimal.**

***Index Terms*—real-time systems; scheduling; energy harvesting; embedded systems;**

I. Introduction and Related work

In this paper, we investigate a real-time system model for embedded systems that collect and store energy from their environment. Such systems are composed, in addition to classical embedded system components, by an energy harvester unit (e.g. a solar panel) and an energy storage unit (e.g. a battery). These harvesting embedded systems are more and more present in our lives: sensor networks in structures such as bridges that collects vibration energy, medical implants that collect energy from the human body, mobile or fix devices with solar panel or windmill etc. Despite of their energy supply particularity, some of these systems need to satisfy strict timing constraints. Therefore, it is important to consider both the time and the energy costs of tasks for the scheduling operations, since both the energy and the CPU times resources of the system have to be shared by the tasks.

The first work addressing the scheduling problem of energy harvesting systems was presented by Mossé in [1]. The problem was solved under a very restrictive task model: the frame-based model where all the tasks have exactly the same period and the same deadline. Later in [2], Moser et al. proposed an optimal algorithm called LSA (Lazy Scheduling algorithm) for periodic and aperiodic tasks. However, in their hypotheses, the CPU frequency can be changed to adjust the Worst Case Execution Time (WCET) of the tasks depending on their energy consumption. Thus, the results of this work rely on the assumption that tasks energy consumption is directly linked to their WCET. A recent work shows that this hypothesis is not suitable for embedded systems [3].

Later, a clairvoyant algorithm called $EDeg$ has been proposed in [4], [5]. The authors expect that this algorithm may be optimal but as far as we know this property has not been proved. The algorithm $EDeg$ relies on a generalizable meta policy: as long as the system can perform without energy failure, a standard policy such as EDF is used. Then, as soon as *future* energy failures is detected, the system is suspended as long as timing constraints are met or until the energy storage unit is full. To detect such future energy failure, the notion of slack time [6] was extended to the notion of slack energy. Another approach based on model checking for dealing with this problem has been proposed in [7].

Recently, in a paper under review process [8], we proposed PFP_{ASAP}, a fixed-priority algorithm that schedules jobs as soon as possible and we proved its optimality for non concrete tasksets.

The aim of this paper is to study the possibility of extending the optimality of PFP_{ASAP} algorithm to concrete tasks by introducing clairvoyance.

The remaining part of the paper is organized as follow. We present the model in Section II. In Section III, we explain why we need a clairvoyant algorithm, then, we propose a new algorithm that we call FPC_{ASAP}. Finally, we discuss future work and we conclude in Section IV.

II. Model

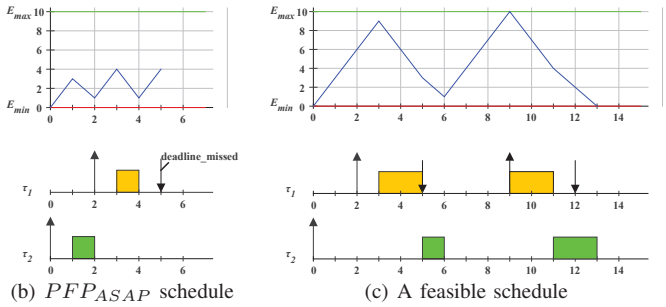
A. Taskset Model

We consider a concrete real-time taskset in a renewable energy environment defined by a set of n periodic and independent tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i is characterized by its priority P_i, its worst case execution time C_i, its period T_i, its deadline D_i, its offset O_i and its worst case energy consumption E_i. The execution time C_i and the energy consumption E_i of a task are fully independent. A task τ_i releases an infinite number of jobs which are separated by T_i

[1] a concrete taskset is a set of real-time tasks whose offsets are known off-line

-	O_i	C_i	E_i	T_i	D_i	P_i
τ_1	2	2	12	10	3	1
τ_2	0	3	15	15	15	2

(a) $E_{max} = 10$, $E_{min} = 0$ and $P_r = 3$



(b) PFP_{ASAP} schedule (c) A feasible schedule

Figure 1. PFP_{ASAP} is not optimal

time units, its j^{th} job is denoted $J_{i,j}$ and must execute during C_i time units and consume E_i energy units. All the tasks consume energy linearly, i.e. a constant amount of energy for each execution time unit. Deadlines are constrained or implicit. The taskset is priority-ordered, task τ_n being the task with the lowest priority.

B. Target Application Description

We consider an embedded system connected to an energy harvesting device. An energy harvesting device is a system collecting energy from its environment (e.g. with a solar panel). The collected energy is stored in an energy storage unit with fixed capacity (e.g. chemical battery or capacitor). We suppose that the quantity of energy that arrives in the storage unit is a function of time which is either known or bounded.

The replenishment of the storage unit is performed continuously even during the execution of tasks, and the energy level of the battery fluctuates between two thresholds E_{min} and E_{max} where E_{max} is the maximum capacity of the storage unit and E_{min} is the minimum energy level that keeps the system running. The difference between these two thresholds is the part of the battery capacity dedicated to tasks execution, denoted C. We suppose that C is as large as needed to never reach E_{max} during replenishment periods. For the sake of clarity, we can consider without loss of generality that $E_{min} = 0$ and that $C = E_{max}$. The battery level at time t is denoted $E(t)$. We note $P_r(t)$ the replenishment function of the battery. Then, in order to simplify the problem, we assume $P_r(t)$ to be a linear function, i.e. $P_r(t) = P_r \times t$, P_r being a positive integer.

Below, we use P_r instead of $P_r(t)$ to denote the replenishment function. The replenishment process in energy harvesting systems is usually slower than the dissipation, for this reason we suppose that tasks consume more energy than the one which is replenished during executions, i.e. $\forall i$, $E_i \geq C_i \times P_r$.

III. The FPC_{ASAP} Algorithm

A. The PFP_{ASAP} algorithm

The PFP_{ASAP} algorithm schedules tasks as soon as possible when there is energy available in the battery, and suspends execution to replenish the energy needed to execute one time unit. A proof of the optimality of this algorithm for non concrete tasksets is proposed in [8] and experiments showed that is has a low overhead. However, the algorithm is no longer optimal when we deal with concrete taskset. Figure 1 shows an example where PFP_{ASAP} fails to schedule a taskset while a valid schedule exists. We can see in Figure 1(b) that PFP_{ASAP} schedules task τ_2 before τ_1 which has no time to replenish enough energy. The PFP_{ASAP} algorithm decides to execute τ_2 without having any information about what will happen later. When τ_1 is activated, the energy available in the storage unit is not sufficient because τ_2 consumed it and the available slack-time is not sufficient to replenish enough energy. However, if PFP_{ASAP} had checked whether the future jobs meet their constraints or not, deadline misses could be avoided, Figure 1(c) illustrates such a decision. This leads us to say that PFP_{ASAP} fails to schedule some feasible tasksets because it does not measure the impact of its decisions on future jobs.

B. As Soon As Possible Clairvoyant Fixed-Priority Algorithm

We aim to find a scheduling algorithm that can be optimal for concrete tasksets. PFP_{ASAP} is not optimal but it can be enhanced to extend its optimality. The algorithm has to be clairvoyant in order to avoid potential future failure or deadline misses.

We propose to study a new clairvoyant algorithm that we call *As Soon As Possible Clairvoyant Fixed-Priority Algorithm* (FPC_{ASAP}). An algorithm is said to be clairvoyant when it takes scheduling decisions according to the future state of the system, i.e. future processor and energy load, future battery level, and the future amount of incoming energy to the storage unit. The clairvoyance must be limited to a finite interval of time that we call *Clairvoyance window*.

The FPC_{ASAP} algorithm inherits the behavior of PFP_{ASAP} and add clairvoyance capabilities. It schedules jobs as soon as possible whenever the two following conditions are met:

- there is enough energy available in the storage unit to execute at least one time unit,
- the execution of the current job does not lead to a deadline miss of jobs of higher priority levels which are requested during clairvoyance window (see below).

If these conditions are not satisfied, the algorithm suspends all executions for one time unit.

Algorithm 1 shows how FPC_{ASAP} takes decisions at time t when a job of priority level i is ready to run. The FPC_{ASAP} algorithm checks first if the execution of jobs of priority level higher or equal to t according to PFP_{ASAP} rules meet theirs deadlines. Algorithm 1 first checks if it is possible to delay the current job by comparing its response time at time $t +$

Algorithm 1 FPC_{ASAP} Algorithm

1: $t \leftarrow 0$
2: **loop**
3: $A \leftarrow$ set of active jobs at time t
4: **if** $A \neq \emptyset$ **then**
5: $J_{i,j} \leftarrow$ the highest priority job of A
6: $d_i(t) \leftarrow$ the next absolute deadline of $J_{i,j}$
7: **if** $ResponseTime_{PFP_{ASAP}}(t+1, J_{i,j}, E(t+1)) > d_i(t)$ **then**
8: execute $J_{i,j}$ for one time unit at time t
9: **else**
10: **if** $Clairvoyance_{PFP_{ASAP}}(t, J_{i,j}, d_i(t), E(t))$ **then**
11: execute $J_{i,j}$ for one time unit at time t
12: **else**
13: suspend the system for one time unit
14: **end if**
15: **end if**
16: **end if**
17: $t \leftarrow t+1$
18: **end loop**

1 with its deadline (line 7), then, it repeats the process for higher priority jobs. This prevent from delaying the current job uselessly, because in the case where it is impossible to delay, if a deadline miss occurs in a higher priority level in the clairvoyance window, the failure cannot be avoided and the system is not schedulable with FPC_{ASAP}.

The first question is: which jobs should be considered in the clairvoyance computation ? In other words, which job can suffer from the energy consumed by the current job of priority level i at time t ?

In fixed-priority context, when a higher priority task is requested, it cannot be delayed by lower priory ones. However, since the energy is a common resource, lower priority tasks can consume the energy needed for higher priority ones and can so delay them. The aim of clairvoyance is to delay lower priority tasks when such a problem is detected. Knowing that a job cannot be delayed more than its deadline, the jobs which are affected by the execution or the delay of $J_{i,j}$ at time t, are the ones of priority levels higher or equal to i which are requested between time t and the absolute deadline of $J_{i,j}$. This defines our clairvoyance window. If a deadline miss occurs after the deadline of $J_{i,j}$, the system is not feasible and clairvoyance cannot avoid that because whatever happens, $J_{i,j}$ must finish executing before its deadline.

The second question is: how can we check if future jobs meet their deadlines ?

A simple way is to compute their response times and compare their termination dates with their absolute deadlines. Since FPC_{ASAP} uses PFP_{ASAP} algorithm for clairvoyance, we have to compute response times according to PFP_{ASAP} rules. Algorithm 2 shows how to compute the response time of a job $J_{i,j}$ at time t with the following notations:

- $x_i(t)$: next activation of task τ_i,
- $we_i(t)$: energy demand of priority level i at time t,
- $wp_i(t)$: processor demand of priority level i at time t,
- $w_i(t)$: time demand of priority level i at time t,
- $c_i(t)$: is the remaining time cost of job $J_{i,j}$ at time t,
- $e_i(t)$: is the remaining energy cost of job $J_{i,j}$ at time t,
- $d_i(t)$: is the absolute deadline of job $J_{i,j}$ at time t.

Algorithm 2 $ResponseTime_{PFP_{ASAP}}$

1: **INPUT** : t, $J_{i,j}$, $E(t)$
2: $w'_i(t) \leftarrow t + \epsilon$
3: **repeat**
4: $w_i(t) \leftarrow w'_i(t)$
5: $we_i(t) = \sum_{l \leq i} \left(e_l(t) + \left\lceil \frac{w_i(t) - x_l(t)}{T_l} \right\rceil \times E_l \right) - E(t)$
6: $wp_i(t) = \sum_{l \leq i} \left(c_l(t) + \left\lceil \frac{w_i(t) - x_l(t)}{T_l} \right\rceil \times C_l \right)$
7: $w'_i(t) = t + \max \left(\left\lceil \frac{we_i(t)}{P_r} \right\rceil, wp_i(t) \right)$
8: **if** $w'_i(t) > d_i(t)$ **then**
9: **return** $w'_i(t)$
10: **end if**
11: **until** $w_i(t) = w'_i(t)$
12: **return** $w_i(t)$

Algorithm 2 behaves like the classical fixed-priority response time algorithm and include tasks energy cost in calculations. It computes analytically the response time of a job $J_{i,j}$ of priority level i by computing progressively the time demand $w'_i(t)$ which is the maximum time needed to satisfy both of the processor demand $wp_i(t)$ and the energy demand $we_i(t)$. The energy demand $we_i(t)$ is derived from the notion of processor demand. It represents the sum of the energy cost of all the jobs of priority equal or higher than i requested during the time interval $[t, w_i(t)[$. Then, the initial battery level $E(t)$ is removed to fit the exact amount of energy to be replenished (Equation is given in line 5 of Algorithm 2).

We deal with the maximum of $we_i(t)$ and $wp_i(t)$ to get the real response time which includes all higher priority interferences and the replenishment periods necessary for a PFP_{ASAP} schedule. Algorithm 2 supposes that the current battery level $E(t)$ and the remaining value of both energy cost $e_i(t)$ and processor cost $c_i(t)$ are known at time t.

C. Clairvoyance computation

At time t, when the FPC_{ASAP} algorithm has to decide whether to execute or not the current job $J_{i,j}$, it uses clairvoyance to predict potential future failure. The clairvoyance consists of computing the response time of all the jobs which are requested during the clairvoyance window, i.e. $[t, d_i(t)[$ and checking if they meet their deadlines. Algorithm 3 shows how to do that.

Algorithm 3 supposes that the remaining cost of tasks and the battery level at each job request are known. This assumption simplifies the clairvoyance computation, however,

Algorithm 3 $Clairvoyance_{PFP_{ASAP}}$ Algorithm

INPUT : $t, J_{i,j}, d_i(t), E(t)$
$R \leftarrow$ set of jobs $J_{k,j}$ of priority level higher than i which are requested during interval $[t, d_i(t)[$
for all $J_{k,j} \in R$ **do**
  $r_{k,j} \leftarrow$ the activation date of $J_{k,j}$
  if $ResponseTime(J_{k,j}, r_{k,j}, E(r_{k,j})) > d_k(r_{k,j})$ **then**
    return $False$
  end if
end for
return $True$

computing these values need heavy calculations and may complexify the clairvoyance algorithm and consequently the overhead of the FPC_{ASAP} Algorithm. This concern remains an open problem.

D. Complexity

The main goal of this work is to provide an optimal algorithm. The complexity of this algorithm is crucial but in this stage of research we first focus on optimality, then, we will study the possibility of reducing the complexity. The computations of clairvoyance and response time algorithms are thus the major keys to the operations performed by FPC_{ASAP} algorithm. The response time of a job at a given instant is given by Algorithm 2 with complexity $O(K.n)$ where n is the number of tasks, and K the number of iterations. In the worst case, K corresponds to the number of interfering jobs and thus is bounded by R/p, where R and p are respectively the longest deadline and the shortest period of tasks. The clairvoyance computation of a periodic taskset at a given time instant can be obtained on-line by computing the PFP_{ASAP} schedule. This is performed with complexity $O(K^2.n)$ by computing the response time of each job activated during the clairvoyance window. The number of these jobs coincides with the number of interfering jobs in the worst case, i.e. K. Therefore, the complexity of FPC_{ASAP} in the worst case is $O(K^2.n)$.

IV. Conclusion and Future work

The aim of this work was to find an optimal algorithm to schedule fixed-priority tasksets in energy-harvesting environment. The PFP_{ASAP} algorithm is optimal but only for non concrete taskset. In this paper we presented FPC_{ASAP} a new algorithm that inherits the behavior of PFP_{ASAP} and add failure predictability capabilities. We explained why we need clairvoyance and we gave a sufficient clairvoyance window length that allow us to verify whether or not there will be future failures caused by early executions.

To continue this work, we plan to explore the following points.

- Optimality: the most important future work is to prove that FPC_{ASAP} is optimal. This algorithm is expected to keep the optimality of PFP_{ASAP} for non concrete tasksets because it behaves the same when there is no future deadline misses and extends it to concrete ones by delaying lower priority jobs as long as needed when a future problem is detected.
- Battery size: in this paper we supposed that the battery is as large as needed to never reach E_{max}. First, we have to find the minimal capacity satisfying this condition. Then, we will study the impact of E_{max} on our algorithm when the battery size is set by the designer.
- Clairvoyance: we will try to find a clairvoyance algorithm that computes the exact response times of jobs with a reasonable complexity.
- Preemptions: we will study the possibility of optimizing the number of preemptions and we will evaluate the performance of a non preemptive version of FPC_{ASAP}.
- Priority assignment: we will study the effect of priority assignment policies on FPC_{ASAP} behavior and clairvoyance complexity, for example reversed Rate Monotonic policy (RM^{-1}) can reduce the number of higher priority interferences which can enhance sensitively the complexity. We will try also to provide a sufficient and necessary feasibility condition that can be used to build an optimal priority assignment (OPA) based on Audsley's algorithm [9] and Davis' criteria [10].
- Assumptions: finally, we will be interested in measuring the effect of the assumptions we set on both replenishment and task consumption functions, indeed, we will try to find the worst case of both consumption and replenishment models.

References

[1] A. Allavena and D. Mossé, "Scheduling of frame-based embedded systems with rechargeable batteries," in *Workshop on Power Management for Real-Time and Embedded Systems (in conjunction with RTAS)*, 2001.

[2] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling with regenerative energy," in *Proceedings of 18th Euromicro Conference on Real-Time Systems*, 2006.

[3] R. Jayaseelan and T. Mitra, "Estimating the Worst-Case Energy Consumption of Embedded Software," in *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006.

[4] H. EL Ghor, M. Chetto, and R. H. Chehade, "A real-time scheduling framework for embedded systems with environmental energy harvesting," *Computers and Electrical Engineering*, vol. 37, pp. 498–510, July 2011.

[5] M. Chetto, D. Masson, and S. Midonnet, "Fixed Priority Scheduling Strategies for Ambient Energy-Harvesting Embedded systems," in *Proceedings of IEEE/ACM International Conference on Green Computing and Communications*, 2011.

[6] J. P. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks fixed priority preemptive systems," in *Proceedings of the 13th IEEE Real-Time Systems Symposium*, 1992.

[7] Y. Abdeddaïm and D. Masson, "Real-time scheduling of energy harvesting embedded systems with timed automata," in *RTCSA*, 2012.

[8] Y. Abdeddaïm, Y. Chandarli, and D. Masson, "The Optimality of PFPasap Algorithm for Fixed-Priority Energy-Harvesting Real-Time Systems," Report, Feb. 2013.

[9] N. Audsley, *Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times*, ser. Technical report. University of York, Department of Computer Science, 1991.

[10] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, 2009.

Notes