RC25685 (WAT1909-033) September 25, 2019
Computer Science

IBM Research Report

Proceedings of the 13th Symposium and Summer School On Service-Oriented Computing (SummerSoc19)

Johanna Barzen[1], Rania Y. Khalaf[2], Frank Leymann[1], Bernhard Mitschang[1]
Editors

[1]University of Stuttgart
Universitätsstraße 38
70569 Stuttgart
Deutschland

[2]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
USA

The 13th Advanced Summer School on Service-Oriented Computing

June 17 - June 23
2019
Hersonissos, Crete, Greece

The 13th advanced Summer School on Service-Oriented Computing (SummerSOC'19) continued a successful series of summer schools that started in 2007, regularly attracting world-class experts in Service-Oriented Computing to present state-of-the-art research during a week-long program organized in several thematic tracks: IoT, formal methods for SOC, Cloud Computing, Data Science, Advanced Manufacturing, Software Architecture, Digital Humanities, Quantum Computing, and emerging topics. The advanced summer school is regularly attended by top researchers from academia and industry as well as by PhD and graduate students.

During the different sessions at SummerSOC renowned researchers gave invited tutorials on subjects from the themes mentioned above. The afternoon sessions were also dedicated to original research contributions in these areas: these contributions have been submitted in advance as papers that had been peer-reviewed. Accepted papers were presented during SummerSOC and during the poster session. Furthermore, PhD students had been invited based on prior submitted and reviewed extended abstracts to present the progress on their theses and to discuss it during poster sessions. Some of these posters have been invited to be extended as a full paper, which are included in this technical report.

Johanna Barzen, Rania Khalaf, Frank Leymann, Bernhard Mitschang
- Editors -

Content

A Pattern-Based Method for Designing IoT Systems

Lukas Reinfurt[1,2], Michael Falkenthal[1], and Frank Leymann[1]

[1] Institute of Architecture of Application Systems, University of Stuttgart,
Universitätsstr. 38, 70569 Stuttgart, Germany
[firstname.lastname]@iaas.uni-stuttgart.de
[2] Daimler AG,
Epplestr. 225, 70546 Stuttgart, Germany
[firstname.lastname]@daimler.com

Abstract. The Internet of Things pattern language can be a valuable tool for practitioners that want to design an IoT system. It offers them abstract proven solutions based on existing real world uses and, thus, makes working with the large amount of different devices, platforms, technologies, and standards in the field of IoT more manageable. Practitioners can use the pattern language to design an IoT system by starting with any pattern they deem suitable and then by continuing to follow the links to related patterns defined by the pattern language. However, when designing an IoT system, applying patterns in a certain order can be beneficial. It allows practitioners to think through important aspects of the system in the right order to minimize context switching and to avoid having to change previous decisions. Thus, we introduce a pattern-based method for designing IoT systems. It guides practitioners through the steps of designing an IoT system in a sensible order. Based on answers to specific questions asked in each step, it points practitioners to suitable patterns and other helpful tools. The result is a pattern-annotated architecture diagram that can be used as basis for further architecture refinement, as a guide for finding existing solutions, and as input for communication with other involved stakeholders.

Keywords: Pattern Languages · Design Patterns · Pattern-Based Method · Internet of Things · System Design

1 Introduction

In recent years, the Internet of Things (IoT) has grown from a vision into reality. More and more devices are capable of gathering and communicating data about their surroundings. Processing this data enables a more detailed understanding of environments and processes, which in turn can be used to control and optimize them. Closing the feedback loop by sending back commands to the devices creates cyber-physical systems that bring benefits into many areas of life, such as smart homes, offices, or factories [20].

But building such systems is not easy. A lot of companies, from small startups to large corporations, are vying to become a household name in the IoT space. New devices, platforms, and technologies are released frequently, creating a vast amount of alternatives that are hard to understand, compare, and reason about. A lack of standardization, or rather a lack of conciliation of the many available standards, has additionally increased the complexity of this situation.

To improve this situation for practitioners who want to understand and build IoT systems, we mined and described IoT patterns in several relevant areas, such as device operation modes and energy supply [27, 28], sensing [28], processing [25], communication [25, 29], registration and bootstrapping [26], management [25, 29], and security [24]. Patterns, as introduced by Alexander in the domain of architecture [2], are structured textual descriptions of reoccurring problems and their abstract solutions. These solutions are based on proven real world implementations and are abstracted in such a way that applying them to a particular problem at hand will result in a slightly different solution every time [2]. But the abstract core of the solution remains the same and is a useful artifact to build up a knowledge base of solutions to common problems and to communicate this knowledge to practitioners.

The IoT patterns are interconnected, for example, if two patterns have to be used together or one after another. Thus, they form a graph, with patterns as nodes, and the relations between the patterns as edges [8]. The semantics of the edges allow practitioners to purposefully navigate through the graph from one pattern to the next. Such a pattern graph is called a pattern language, which is a useful tool for practitioners when trying to understand a particular domain or when trying to build something in that domain. For the IoT domain, we authored the IoT pattern language[3], which combines all our IoT patterns. A practitioner can use the IoT pattern language to solve problems in his IoT system. He can start with one particular problem, apply the appropriate pattern, and then follow the links between the patterns to solve additional problems.

But there is always the question of where to start. If the practitioner has an existing system where he wants to improve some aspects, he can use an approach to find suitable entry points for his current situation based on the problems he currently has [30]. But if the practitioner is starting from scratch, he might need a more guided approach to build an IoT system using patterns, which allows him to think through the important aspects of the system one at a time and in a sensible order. By concentrating on one aspect at a time, context switches are minimized and related decision are made at the same time. By prescribing a sensible order, problems, where more recent decisions change the outcome of past decisions, can be minimized. In this paper, we describe such an approach for the IoT patterns.

The remainder of the paper is structured as follows: We motivate our approach in Section 2. Related work is presented in Section 3. Our approach in general and details about each step are described in Section 4. This section also presents a running example. Section 5 concludes this paper and presents possible future work.

2 Motivation

For our examples throughout this paper we are going to focus on a concrete practitioner, Patrick, shown in Figure 1. He is an IoT system architect working as a consultant for private clients. In his projects, Patrick is usually tasked with helping these clients with their IoT systems. Sometimes these systems already exist and have to be modified or fixed. But now, Patrick has the task to create a home automation system from scratch.

[3] `http://www.internetofthingspatterns.com`

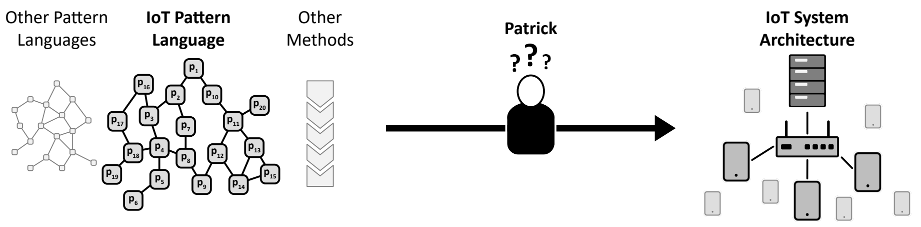

Fig. 1. The practitioner Patrick wonders how he should start designing a new IoT system.

Patrick knows that designing and implementing IoT systems is complex. In the last few years, he has seen the countless devices, services, and platforms that have been introduced, using a large number of different technologies and standards. Some of these existed before and have been adapted for the new challenges of IoT systems, while others are new and were specifically designed with these challenges in mind. This has created a vast and complex landscape of IoT solutions and technologies that is hard to understand. Thus, Patrick is looking for tools and methods that can help him handle this complexity.

But not everything in the IoT is new. Many components and technologies that are used in IoT systems have existed and have been used independently. For example, the backend of an IoT system is often located in the cloud, which is a computing paradigm that has been very successful independently from the IoT. Or when it comes to security, many techniques are used that have been established in IT systems long ago. In these areas, there also exist many tools that help practitioners like Patrick to design IT systems with best practices in mind, such as the Cloud Computing Patterns [11] or various patterns for security [13, 32, 33].

However, the IoT adds a new dimension in many areas. It introduces physical devices that are often constrained in their capabilities. These constraints influence all other areas that are involved in IoT systems, from communication networks, to processing, management, and security. Some of the existing tools are still applicable, but new approaches that take the characteristics of IoT systems into account are also required. To offer practitioners, like Patrick, tools to help them with understanding and designing IoT systems, we introduced the IoT pattern language, which contains patterns for various aspects of IoT systems.

Patrick has found the IoT pattern language and wants to use it to design the smart home system for his client. He could start with some patterns and then follow the links to related patterns that are defined in the pattern language. This may lead to patterns of different areas being used in a mixed order, which would require Patrick to switch between those contexts and may lead him to having to revise old decisions when a newly applied pattern introduces new influencing factors. Instead, it makes more sense to finish one area first before moving to the next and to move through these areas in a sensible order. Another problem is that the IoT pattern language is not exhaustive. It links to patterns from other pattern languages where applicable, but it also makes sense to use other tools during the system design process that are not pattern related.

We think that a more guided approach for designing IoT systems from scratch can help practitioners like Patrick. Therefore, we introduce a pattern-based method for designing IoT systems in the following sections. It not only guides practitioners through a sensible application order of the IoT patterns and other related patterns, but also incorporates or points to other useful tools. Thus, it offers a systematic process for the pattern-based design of IoT systems.

3 Related Work

IoT systems are essentially cyber-physical systems, for which a number of design and modeling approaches have been described in literature [15, 18]. But these do not incorporate patterns. A pattern-based approach for designing reliable cyber-physical systems was presented in [22], but it is focused on so called reliable component composition patterns that can improve the reliability of components by serializing or parallelizing them, and have little in common with the patterns as we use them. To the best of our knowledge, there is no comprehensive pattern-based method for designing IoT systems described in literature.

But for some parts of IoT systems, pattern-based methods exist. For example, the Cloud Computing Patterns [11] are applicable in IoT systems where components, usually in the backend, are run on a cloud infrastructure. The pattern-based design method for cloud applications that uses these patterns [9, 10] can be helpful when designing an IoT system backend and can be integrated in our method. But the patterns and method do not help with the IoT specific problems of small, constrained devices and their effects.

Enterprise integration patterns are also applicable in IoT systems. They are clustered by six root patterns, each describing a part of a messaging pipeline and pointing to more detailed patterns for its implementation [16]. This allows a practitioner to design a messaging pipeline from beginning to end by following the patterns, which resembles the approach presented in this paper. But these patterns are limited to communication and, thus, are only useful for designing parts of an IoT system. They can be integrated in the *Communication* step of our method.

Patterns for designing distributed control systems [4] may also be applicable to some IoT systems, but they also lack solutions for IoT specific problems of small, constrained devices. They also do not offer a method, but only describe a design approach based on following the links between patterns in a pattern language once problems in an existing design or system become apparent [4].

Another area where existing patterns are applicable is security. Several general methodologies for adding security to distributed systems exist [35]. When it comes to patterns, *Security Patterns in Practice* [13] describes three different approaches for pattern-based secure systems development. In the first approach, information about their use is added to the patterns themselves to offer application guidance to practitioners. The second approach is life-cycle-based, where security patterns are applied at every development stage. The third approach uses model-driven engineering, where models are transformed from stage to stage according to rules and meta-models [13]. A different architectural systems engineering methodology for addressing cyber security with patterns is also presented in [3]. Depending on their scope, these different methodologies and approaches can either be integrated into our method at the *Security* step, or can themselves integrate our method where appropriate. But these methods and pattern only handle some security aspects of IoT systems and, thus, these approaches do not cover other IoT specific problems.

There are other pattern languages that are not applicable when designing IoT systems but follow a similar approach as the approach presented in this paper. *Designing Interfaces* [34] presents patterns for user interface design. They are clustered in groups that are ordered by scale, which also approximates the order of the design process. A practitioner starts designing the biggest user interface pieces first and then gradually adds smaller and smaller details. *SecPatt* [1] focuses on easy to use security patterns for web sites. The patterns are clustered in terms of the practitioner's experience. As a beginner, a practitioner starts with the first pattern cluster. As he gains more experience, he can move to patterns in more advanced clusters. What these approaches have in common with our proposed approach is that they group their patterns into chronologically ordered clusters that guide practitioners through a sensible application process. But they do not use guiding questions to further add structure inside these clusters or to integrate other methods and tools.

On a more abstract level, the graph created by the IoT pattern language through the links between its patterns can be seen as a *dense primary index structure* [12]. The steps of the method presented in this paper would then create *sparse secondary index structures* (views) around context clusters [12], while the overall method orders these views chronologically. The notion of temporal ordering of pattern application is also at the center of *pattern sequences* [23]. It is often necessary to compose patterns from different domains to create a system, and the order of application matters. A pattern sequence creates a temporal order of patterns of one or multiple domains. In addition, a pattern sequence is also more effective in presenting a pattern language. Compared to a pattern language graph, a sequence provides a linear order in which the patterns can be listed [23]. The method presented in this paper can be seen as the main sequence of the IoT patterns and related patterns. Another approach is to use patters to capture architectural decisions, as they do not only describe the decisions themselves, but also contain possible alternatives, consequences, and the rational behind the decisions. This can be useful, as these decisions can not be derived from architecture diagrams and would be lost if they are not recorded otherwise [14]. This approach can be combined with domain specific *Reusable Architectural Decision Models* (RADMs) into a method, called *ArchPad* that guides pattern selection and architectural decisions [37]. The method presented in this paper can be seen as a *RADM for IoT systems* that concentrates on stage 2 (conceptual decisions) and 3 (technology decisions) of the ArchPad method.

Overall, we think that other approaches have shown that pattern-based methods can add value to pattern languages. The lack of a comprehensive method for the IoT domain leaves room for a method based on the IoT pattern language, which can also integrate other existing approaches at the appropriate stages. The next section present our proposal for such a method.

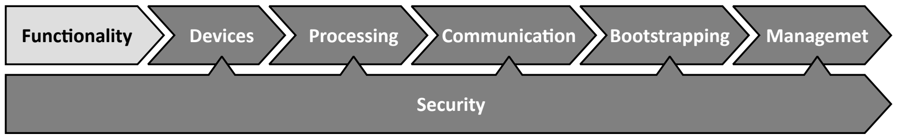

Fig. 2. An overview of the pattern-based method for designing IoT systems and its main steps.

4 Method

Our proposed method is a seven step process based on major pattern clusters in the IoT pattern language. Figure 2 shows the order of these steps. Note that, while the general order of the steps is linear, it might make sense to track back or iterate at some points, but this is not further shown here. In each step, targeted questions about specific details are asked. Depending on the answers, the method then points to patterns from the IoT pattern language or from other pattern languages, suggests useful tools or methods, or describes modifications to the architecture diagram. By following the method, practitioners can design an IoT system architecture that is annotated with patterns.

The method, as described here, uses Unified Modeling Language (UML) [21] activity diagrams to represent the architecture. This modeling language was chosen because it is standardized, widely used, and supports all constructs necessary to describe the structure, as well as the functionality, of an IoT system. Other modeling languages can be used instead if required. Each step of the method and its questions address certain quality attributes, such as *functional suitability*, *performance efficiency*, *compatibility*, *usability*, *reliability*, *security*, *maintainability*, *portability*, and their sub-attributes, as defined in *ISO/IEC 25010:2011* [17]. These quality attributes are mentioned where appropriate, but a more detailed description of how each pattern influences those attributes can be found in the pattern text. Note, that the patterns may not address all possible quality attributes and, thus, further work might be necessary to address those that are missing.

The first step, *Functionality*, is not supported by patterns, but is a vital precursor for the following steps. Here, the overall goals of the system and all activities necessary to reach those goals are collected. Additionally, important parameters, like physical location and the estimated size of the collected measurements are established, which inform the pattern selection in later steps. Crucially, the first step also decides if the planned project is actually an IoT project and if the later steps are applicable at all. If so, the following six steps can be used to design an IoT system architecture based largely on the IoT pattern language. Thus, this step mainly addresses *functional suitability* and *time behavior* of the system, as well as *capacity* on a data level.

In the *Devices* step, the capabilities and functionality of the devices are detailed, using the IoT patterns for energy supply, operation mode, and sensing [27, 28]. It addresses questions of *functional suitability*, *performance efficiency*, *compatibility*, and *reliability* on a device level.

Based on the capabilities of the devices, the *Processing* step defines where, when, and how the data they collect should be processed. In some cases this requires the introduction of new components into the architecture, which is, thus, crucial in shaping the component landscape of the planned IoT system. This step addresses *functional suitability*, *performance efficiency*, and partly *usability* of processing components.

In the next step, *Communication* between all these components is specified in more detail. This involves estimating the required communication bandwidth, and thinking about how and when communication should happen and how reliable it should be. Thus, this step addresses *functional suitability*, *performance efficiency*, *compatibility*, and *reliability* of communication.

Once all communication paths are defined, there is one more step to make the system operational. All devices and other components have to be initially configured to be able to integrate into the system and start communicating with others. How this is done is defined during the *Bootstrapping* step. Here, IoT patterns are used to describe i) how to initially place bootstrap information onto devices, ii) how to register devices in the system, and iii) how to define and store the registration information. Thus, this step is mainly concerned with *portability* and its sub-characteristics of *adaptability* and *installability* of the devices in the system.

The system is now functional, but there are two more steps in the method. In the *Management* step, processes for updating software and firmware on devices, and how to organize and scale this for large numbers of devices, are described. This is crucial to ensure the longevity of the system, as it allows practitioners to keep components up to date and in line with changing requirements. Thus, this step is concerned with questions of *maintainability*, mainly its sub-characteristic *modifiability*, of the system components.

Underlying all of the previous steps is *Security*, where methods for securing devices, processing, communication, bootstrapping, and management are described. It is executed in parallel with the previous steps in order to ensure that security concerns are considered at the right times during the method, as some security decision might also influence decisions in other steps. This is again a crucial step, as it hardens the system against potential security risks. Thus, this step is concerned with questions of *security* and its sub-characteristics of *confidentiality*, *integrity*, and *authenticity*, of the entire system.

After completing these seven steps, a practitioner now has an architecture diagram that specifies all major components of the systems, as well as their functionalities and their communication paths. These are annotated with patterns that describe some of their aspects in more detail. At this state, the practitioner can use this diagram in several ways.

First, he can use the component descriptions to take a look at existing solutions that he might want to use in his system. Since the diagram defines i) required functionality, ii) behavior, iii) communication, and iv) bandwidth requirements, he now has a solid understanding of what a particular solution has to offer to fit into the system. This should enable him to compare available offerings more easily and to communicate his requirements more effectively to vendors.

But he might not be able to find existing solutions for all components. This leaves him with some components of the system that have to be implemented from scratch. He can use the architecture diagram as a starting point for a more detailed specification process for these components. The information in the diagram should give him a good idea of the requirements of those components, and the annotated patterns offer ways to implement their functionality.

The practitioner can also use the architecture diagram when communicating with other stakeholders. As IoT systems often span multiple locations and areas of responsibilities, the diagram can be helpful in i) finding out what exactly these areas are in order to find the right people to talk to, and for ii) showing different stakeholders what and why something is required from them.

4.1 Functionality

The whole point of an IoT system is to support a certain functionality that is routed in data collected and processed by devices and other components. Thus, you should first define the goals of the system and what data is needed to reach those goals. In this method, this is done be answering the following questions, which will point to appropriate patterns and other tools where applicable. Figure 3 shows an exemplary result of applying the *Functionality* step of the method. It is explained in more detail by the application example at the end of this section.

Questions

Main Functionality - What do you want to accomplish? First, you need to be aware of your *main functionality*. These are the things that your system should ultimately accomplish if it has access to the right data. An example would be "display the average temperature of locations A, B, and C", or "automatically turn on the light if someone enters the room". We draw these as *StructuredActivityNodes* at the top of our canvas. This allows us to fill in detailed actions, decisions, splits, and joints later on if required.

Prerequisite Functionality - What has to be done before? Usually there is some *prerequisite functionality* that needs to happen before the *main functionality*. This can be worked out by planning in reverse starting from the *main functionality* nodes. First, for each *main functionality*, note its parameters. For each parameter, ask yourself what *prerequisite functionality* delivers this parameter. Draw the *prerequisit functionality* under their corresponding *main functionality*, put an *ObjectNode* representing the corresponding parameter between them, and connect everything through *ObjectFlows*. Then, for each newly added functionality, repeat the above process to get its *prerequisite functionality*, until no more *prerequisite functionality* can be found. This results in a graph, where the vertices that generate data or where commands terminate represent the *root functionality*. Vertices in between the *root functionality* represent *intermediary functionality* that combines, splits, or decides on its incoming data through some form of processing (see Section 4.3). The *ObjectNodes* passed between the functionality nodes represent the data that is required to ultimately execute the *main functionality*.

Postrequisite Functionality - What has to be done after? Usually there is also some *postrequisite functionality* that is triggered by the *main functionality*. Repeat the process for *prerequisite functionality*, but think about what has to happen after the *main functionality*. Iterate until no more *postrequisite functionality* can be found. Again, you might get *intermediary functionality* that, in this case, splits the results of previous functionality through some form of processing (see Section 4.3).

Origin and Target - Where does functionality originate and terminate? For each *root functionality*, ask yourself which entity would execute it. This can be a person, a workflow, an API, a device, etc. Add the origin for each *root functionality* as a *PartitionName* in parentheses above the functionality. If you are left with a set of *root functionality* nodes that do not include a device *partition*, then you do not have an IoT project and the remainder of this method may be of little value to you.

Physical Location - Where does functionality happen? For each functionality, ask yourself where it will be executed physically. If you end up with more than one physical location, use *ActivityPartitions* in the form of vertical swimlanes to organize all functionality. Use the *«external» ActivityPartition* for functionality outside your control.

Functionality Timing - How often is functionality executed? Depending on how time critical your *main functionality* is, you might need to run the *pre-* and *postrequisite functionality* in shorter or longer intervals. It might also be necessary to execute this functionality whenever a certain event occurs. Traverse the graph breadth-firsts, starting with the *main functionality* nodes, and think about how frequently you need each of them executed. Add a *TimeEvent* (in seconds) or a *ReceiveSignal* to the respective nodes. Then, for each *pre-* and *postrequisite functionality*, add the smallest *TimeEvent* of all direct parents as this functionality's *TimeEvent*. If the parents have *ReceiveSignals*, add corresponding *SendSignals*. Continue this process for each functionality until you have reached the bottom layer of *root functionality*.

Object Size - How big are the parameter objects? The size of the parameter objects that are passed between the functionality nodes can vary. Together with the *functionality timing*, the *object size* is useful to estimate the communication and processing bandwidth required in later steps. Traverse the graph breadth-first, starting from the *root functionality* nodes, and estimate the upper bound of the size of all the parameter objects they produce. Annotate each *ObjectNode* with its estimated size in bits.

Application Example

We now come back to Patrick, the IoT system architect tasked with building a smart home system for one of his clients. Patrick starts with our method by working through the questions of the first step, the result of which is shown in Figure 3. The *main functionality* of this system is *Turn on Light If Sunset and Motion* is detected, *Turn On Light If Button Pressed*, and *Log Events*. The *prerequisite functionality* that has to happen before them are *Sense Press*, *Sense Sunset*, and *Detect Motion*. *Sense Press* originates from a devices placed inside the house, while *Sense Sunset* comes from an external weather API. *Detect Motion* has itself two *prerequisite functionality* nodes, *Sense Motion*, since the driveway is too large to be observed by one motion sensor. These also originate from devices that have to be located on the driveway. There is only one *postrequisite functionality*, *Turn On*, which happens after either *Turn on Light If Sunset and Motion* or *Turn On Light If Button Pressed*. This functionality targets a device on the driveway. The *ObjectFlows* between these functionality nodes show the messages that have to be passed around.

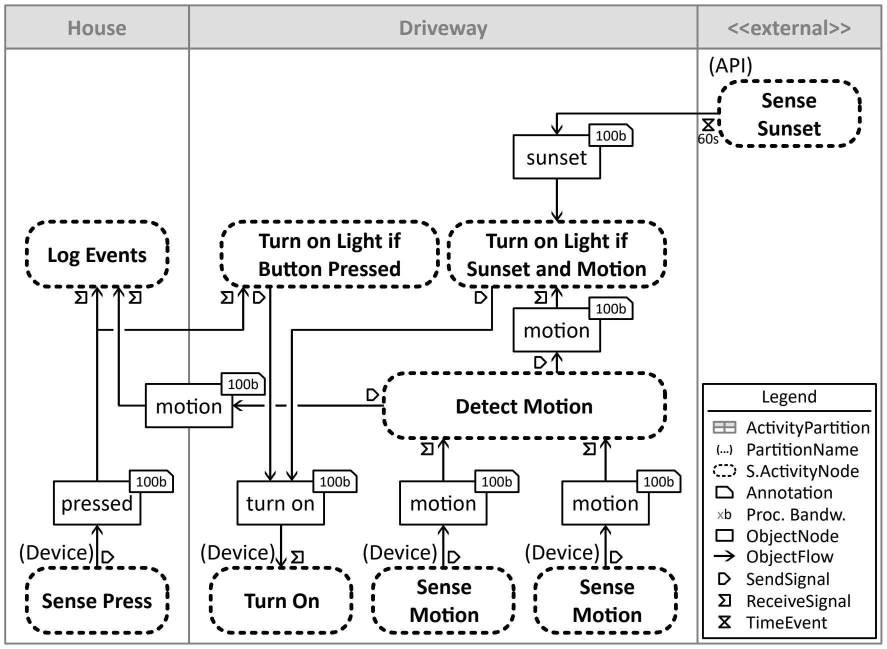

Fig. 3. Exemplary IoT system architecture after *Functionality* step.

Each one is annotated with a *functionality timing* - a 60 seconds *TimeEvent* for *Sense Sunset* and *SendSignals* for all others. The size of the objects passed around is estimated at 100 bits for a JSON object with an ID, payload, and timestamp. This first step provides a rough overview of the systems functionality and how it is connected.

4.2 Devices

Devices are usually the main data sources in IoT systems and can vary significantly in size, capabilities, energy supply, or mobility, depending on the use case. In this step, the number of devices in the system and their capabilities are defined in more detail. Figure 4 shows an exemplary result of applying the *Devices* step of the method. It is explained in more detail by the application example at the end of this section.

Questions

Device Reuse - Do you have existing devices you could use? For each device *partition*, ask yourself if you already have existing devices that support the required functionality. It might be possible that you can cover all or some of your *root functionality* with devices that you already operate or own. Change the corresponding *PartitionNames* to known devices where applicable.

Device Amount - How many devices do you need? In some situations you might need a separate device for every *root functionality* node. But often, one device can support multiple *root functionality*. For example, if your *root functionality* includes *measure temperature* and *measure humidity*, you can find many devices that offer both in one package. Combining devices where possible decreases costs and complexity. If possible, combine device *partitions* and merge their update intervals by selecting the smallest *TimeEvent* and combining all *SendSignals* and *ReceiveSignals*.

Energy Supply - How will you power the devices? How you power the devices depends on their *functionality timing* and *physical location*. Use a Mains-Powered Device [28] if mains power is readily available at the device's *physical location* or if its *functionality timing* is very short. Otherwise, especially if the device is mobile, select one of the following power supply types: If the device's *functionality timing* is still pretty short, a Period Energy-Limited Device [27] might make sense. If the device has to be rugged or will be operated in a hard to reach *physical location*, and if its *functionality timing* is rather long, use a Lifetime Energy-Limited Device [28]. If the device's *functionality timing* is rather low and its *physical location* is suitable, use an Energy-Harvesting Device [27]. Annotate all devices with the appropriate energy supply patterns.

Operation Mode - How will the devices operate? A device's operation mode is largely dictated by its *functionality timing* and energy supply. Mains-Powered Devices [28] or devices with very short *functionality timing* are usually Always-On Devices [28]. Period Energy-Limited Device [27], Lifetime Energy-Limited Device [28], Energy-Harvesting Device [27], or devices with long *functionality timing* are usually Normally-Sleeping Devices [27]. Annotate all devices with the appropriate operation mode patterns.

Sensing - How will the devices gather data? The kind of sensing that is needed on a particular device is a result of its *functionality* and *functionality timing*. If it only uses *TimeEvents*, Schedule-Based Sensing [28] makes sense. If it only uses *SendSignals*, Event-Based Sensing [28] makes more sense. If both are required, a combination of both sensing paradigms is appropriate. Annotate all devices with the appropriate sensing patterns.

Secure Devices - How will you secure devices? Consider Section 4.7 in regards to securing devices. Make sure that the previous decisions support the required security measurements and adjust if necessary.

Application Example

Based on the results of the first step, Patrick is now able to continue with the second step of our method, concentrating on the *Devices* shown at the bottom of Figure 4. The client actually owns two devices that can be reused for this system, a *Flic* button for the *Sense Press* functionality, and a *Philips Hue* light bulb for the *Turn On* functionality. The devices for the *Sense Motion* functionality have to be added to the system. Combining

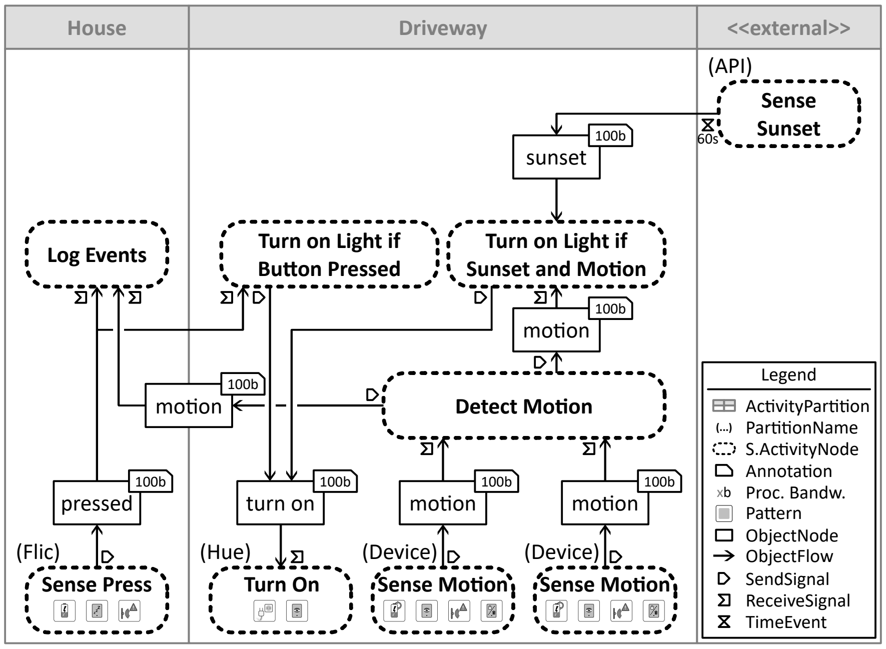

Fig. 4. Exemplary IoT system architecture after *Devices* step.

devices is not possible in this scenario. For each device, Patrick now specifies its energy supply, operation mode, and sensing capabilities, based on our IoT patterns. So, for example, the *Philips Hue* light bulb is a Mains-Powered Device and a Always-On Device. Regarding device security, as most device are located on the driveway, where they may be prone to theft, Patrick adds the Remote Lock and Wipe pattern to them where possible. The following table lists all patterns that have been applied in this step.

Entity	Patterns
Sense Press *StructuredActivityNode*	Lifetime Energy-Limited Device, Normally-Sleeping Device, Event-Based Sensing
Turn On *StructuredActivityNode*	Mains-Powered Device, Always-On Device
Sense Motion *StructuredActivityNode*	Period Energy-Limited Device, Normally-Sleeping Device, Event-Based Sensing, Remote Lock and Wipe

4.3 Processing

Once the functionality of the system and the required data is defined, the next question is what has to happen to the data so that the functionality can be implemented. An IoT system usually includes some processing components, since the data is rarely in exactly the form that it needs to be in. Figure 5 shows an exemplary result of applying the *Processing* step of the method. It is explained in more detail by the application example at the end of this section.

Questions

Processing Bandwidth - How much data has to be processed? Starting from the device *partitions*, for each parent functionality, calculate the sum of all of its *ObjectNode* inputs in bits per second. This number is how much data has to be processed per second to perform the functionality. Annotate this number to each *StructuredActivityNode* as *processing bandwidth*. Continue upwards until all *StructuredActivityNodes* are annotated.

Processing Location - Where should the processing be done? Certain functionality might be computed through LOCAL PROCESSING right on or near the devices. If a functionality that is not already part of a device requires only input data from a single device, and if the *processing bandwidth* needed to execute this functionality is sufficiently small, then LOCAL PROCESSING right on the device is an option. In this case, move the functionality into the *StructuredActivityNode* of the device. Place this device into a horizontal swimlane called *Local* and add the LOCAL PROCESSING pattern to it. If a functionality requires input data from multiple devices and if the required *processing bandwidth* is reasonable, then LOCAL PROCESSING on a component physically near the device can offer lower latencies. In this case, add a new *StructuredActivityNode* with the component's name as *PartitionName* and move this functionality into it. Place this *StructuredActivityNode* into a horizontal swimlane called *Edge*, which is located above the *Local* swimlane, and add the LOCAL PROCESSING pattern to it. Otherwise, if the required *processing bandwidth* is too high, use REMOTE PROCESSING to execute this functionality in the cloud or on a backend server. Add a *PartitionName* with the corresponding cloud service or backend server. Place this functionality in a horizontal swimlane called *Remote*, which is located above the *Edge* swimlane, and add the REMOTE PROCESSING pattern to it. Also check, if this functionality is still in the right *physical location* swimlane, or if it has to be moved.

Processing Time - When should the processing be done? For each functionality, look at its *functionality timing*. If there are only *ReceiveSignals*, use EVENT-BASED PROCESSING. If there are only *TimeEvents*, use SCHEDULE-BASED PROCESSING. If both are present, it might be necessary to use two separate processing pipelines, one for EVENT-BASED PROCESSING and one for SCHEDULE-BASED PROCESSING. It might also be possible to use EVENT-BASED PROCESSING to emulate SCHEDULE-BASED PROCESSING. BATCH PROCESSING can be useful to optimize resource utilization if low latency is not required. Annotate all functionality nodes with the appropriate pattern.

Processing Implementation - How do you implement processing steps? If you require the flexibility that processing can be changed and adjusted on the fly, even by non-programmers, a RULES ENGINE [25] can be helpful. For each functionality, ask yourself if this functionality is needed here. If so, annotate this functionality with the RULES ENGINE [25] pattern.

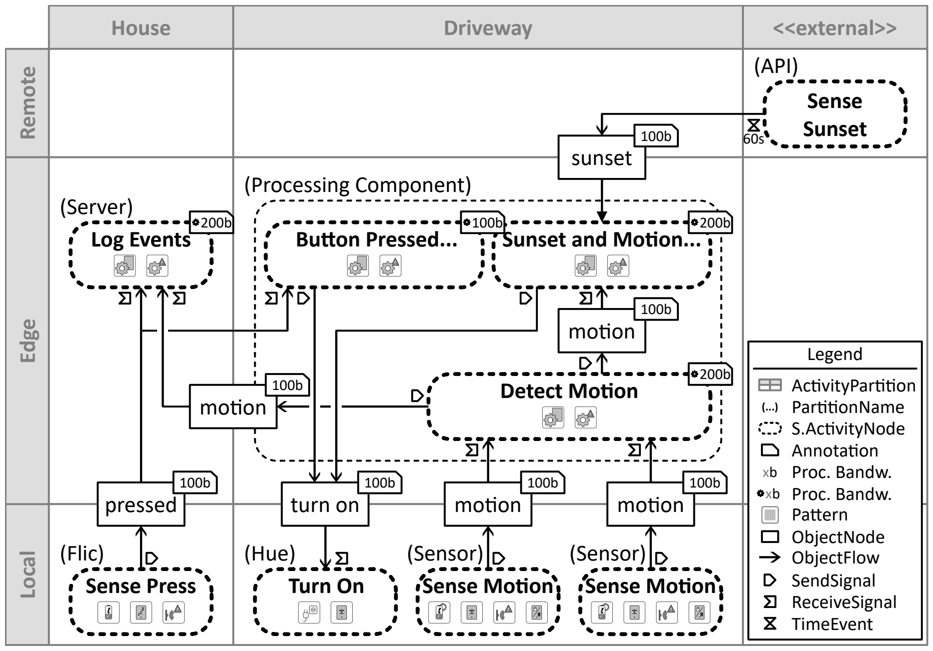

Fig. 5. Exemplary IoT system architecture after *Processing* step.

Combined Processing - Can processing functionality be combined? After the previous steps you might end up with multiple processing components in the *Edge* or *Remote* swimlane. For each swimlane, think about if any of these components can be physically combined. Maybe some of them do very similar processing, or maybe multiple processing functionality nodes can be executed by one shared RULES ENGINE [25]. If so, merge the *StructuredActivityNodes* of the involved functionality and give the new node a suitable *PartitionName*. Merge their update intervals by selecting the smallest *TimeEvent* and combining all *SendSignals* and *ReceiveSignals*.

Secure Processing - How will you secure processing? Consider Section 4.3 in regards to secure processing. Make sure that the previous decisions support the required security measurements and adjust if necessary.

Additional Resources

Often, processing functionality can be resource intensive and has to be executed remotely. The cloud is a natural fit for such tasks. When designing components in the *Remote* layer, the Cloud Computing Patterns can be helpful [11]. The pattern-based design method for cloud applications [9, 10] can also be used in this step.

Application Example

In this step, Patrick is now able to concentrate on *Processing*. The estimated processing bandwidth of each *intermediary functionality* is calculated based on the previously estimated *functionality timings* and *object sizes*. Based on these figures, Patrick decides that Local Processing is feasible. But because the existing devices do not have processing capabilities, the *intermediary functionality* has to be processed on separate components, which places them in the *Edge* swimlane. Event-Based Processing is chosen, as communication is largely event-based. No special processing implementation is required. But Patrick decides that all *intermediary functionality* except *Log Events* can be combined on one edge device to save costs and, thus, combines them in a *Processing Component* that will be placed on the driveway near the sensors to reduce latency. *Log Events* will be executed on a small server in the house, where more storage space for the logs is available. He does not add extra security measures for processing, as no critical data or processing is part of the system. The result of this step is shown in Figure 5. The following table lists all patterns that have been applied in this step.

Entity	Patterns
Log Events *StructuredActivityNode*	Local Processing, Event-Based Processing
Button Pressed... *StructuredActivityNode*	Local Processing, Event-Based Processing
Sunset and Motion... *StructuredActivityNode*	Local Processing, Event-Based Processing
Detect Motion *StructuredActivityNode*	Local Processing, Event-Based Processing

4.4 Communication

In the previous steps, we introduced several *StructuredActivityNodes* that represent separate components. These components are further separated into several *physical locations*, as well as the *Local*, *Edge*, and *Remote* layers. Once the physical separation of these components is clear, it is necessary to think about communication between these components. Figure 6 shows an exemplary result of applying the *Communication* step of the method. It is explained in more detail by the application example at the end of this section.

Questions

Communication Partners - Who has to communicate? The graph now consists of three different kinds of larger nodes that inform all communication. You have local devices, some of them with integrated Local Processing. There might be some additional Local Processing nodes in the *Edge* layer. The remainder are the Remote Processing nodes in the *Remote* layer. Their communication paths are indicated by the *ObjectFlow* edges between these nodes. If some of these nodes do not support the same communication technology as their communication partners, a Device Gateway [25] between them might be necessary. If so, add a new *StructuredActivityNode* between them in the *Edge* layer and annotate the Device Gateway [25] pattern to it. Check if any Local Processing in the *Edge* layer can be done on this Device Gateway [25] to reduce the number of components in the system.

Communication Bandwidth - How much bandwidth is required to communicate? For each *intermediary functionality*, the required *communication bandwidth* can be calculated from its incoming and outgoing *ObjectFlows*. Its required download bandwidth equals the sum of all its incoming *ObjectFlows* in bits per second. Its required upload bandwidth equals the sum of all its outgoing *ObjectFlows* in bits per second. Check if these numbers are reasonable or if you might have to split up nodes. Delta Updates [29] can also be used if communication bandwidth is limited.

Communication Type - How should be communicated? For Period Energy-Limited Devices [27], Lifetime Energy-Limited Devices [28], or Energy-Harvesting Devices [27], Low-Power Short-Range Communication or Low-Power Long-Range Communication might be used, depending on the distance to their next communication partner. Energy-Harvesting Devices [27] or devices with a really small energy supply might also be able to use Passive Communication. If devices have to communicate between themselves, a Mesh Network might make sense. Mains-Powered Devices [28] can use Powerline Communication to reduce infrastructure costs. If wireless communication is difficult and wired communication is not possible, Visible Light Communication [29] can be an alternative. Annotate all *ObjectFlows* with suitable communication type patterns.

Communication Time - When should be communicated? For each *ObjectFlow*, the communication time is informed by the *functionality timing* of the sender. For *TimeEvents*, use Schedule-Triggered Communication. For *SendSignals*, use Event-Triggered Communication. In some case, Command-Triggered Communication can be an alternative. If *TimeEvents* are very short, or if *SendSignals* tend to happen very close to each other, Batched Communication can be helpful to avoid overwhelming devices and networks. In some cases, it might make sense to send newly connected communication partners parts of previous communication exchanges as Pre-Subscription Notification to get them initialized. Annotate all *ObjectFlows* with suitable communication time patterns.

Communication Reliability - How reliable should communication be? A Device Shadow [25] decouples communication between Normally-Sleeping Devices [27] and other components. A Device Wakeup Trigger can be used to wake up Normally-Sleeping Devices [27] when necessary. Fire-and-Forget Telemetry can be used to send data where high reliability is not required. Reliable Telecommand is better suited for messages, like commands, that should reach their intended recipient. Queued Messages can be used to store messages for communication partners that are currently offline or busy. Once they reconnect, it is possible to Send All Pending message or

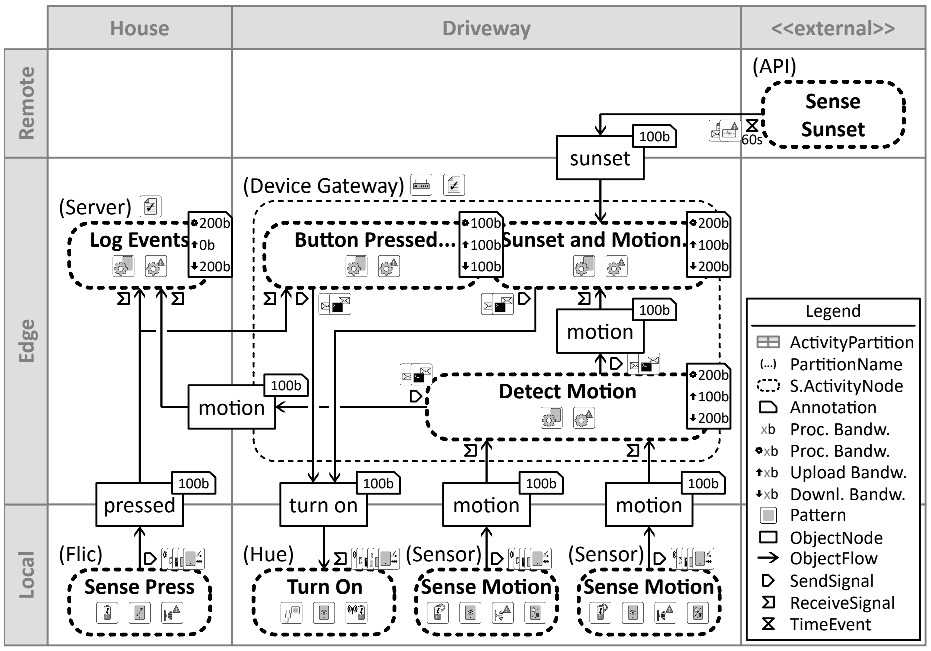

Fig. 6. Exemplary IoT system architecture after *Communication* step.

only Send Latest. Use Rate Limiting to prevent large numbers of messages from overwhelming the network and devices. Use a Repeater to ensure reliable communication over larger distances. Annotate all *ObjectFlows* and *StructuredActivityNodes* with suitable communication reliability patterns.

Secure Communication - How will you secure communication? Consider Section 4.7 in regards to securing communication. Make sure that the previous decisions support the required security measurements and adjust if necessary.

Additional Resources

In addition to the patterns mentioned above, the Enterprise Integration Patterns [16] are useful when using messaging-based communication.

Application Example

Now that Patrick has a clear idea where data is generated, used, and processed, he can continue with the *Communication* step. The communication partners are already clear from the previous steps, but the *Philips Hue* light bulb requires a Device Gateway to be able to communicate with other components. Patrick adds a Device Gateway and decides that it should also handle the Local Processing in the driveway. He estimates and annotates the *communication bandwidth* required between these components based on the previous estimates for *functionality timings* and *object sizes*. He uses IoT patterns to describe the *communication type* and *time* of the components. For example, the *Flic* button uses Low-Power Short-Range Communication and Event-Based Communication. All messages should reach their intended target eventually, so he decides to also use Reliable Telecommand. He also wants to secure communication with Trusted Communication Partner and Outbound-Only Connection on the devices, a Whitelist on the server and Device Gateway, and Asymmetric Encryption [13] on all communication channels. The result of this step is shown in Figure 6. The following table lists all patterns that have been applied in this step.

Entity	Patterns
Server *PartitionName*	Whitelist
Device Gateway *PartitionName*	Device Gateway, Whitelist
sunset *ObjectFlow*	Schedule-Triggered Communication, Fire-and-Forget Telemetry, Asymmetric Encryption
motion from **Detect Motion** *ObjectFlow*	Event-Triggered Communication, Reliable Telecommand, Asymmetric Encryption
pressed *ObjectFlow*	Low-Power Short-Range Communication, Event-Triggered Communication, Reliable Telecommand, Trusted Communication Partner, Outbound-Only Connection, Asymmetric Encryption
turn on *ObjectFlow*	Low-Power Short-Range Communication, Event-Triggered Communication, Reliable Telecommand, Pre-Subscription Notification, Trusted Communication Partner, Outbound-Only Connection, Asymmetric Encryption
motion from **Sense Motion** *ObjectFlow*	Low-Power Short-Range Communication, Event-Triggered Communication, Reliable Telecommand, Trusted Communication Partner, Outbound-Only Connection, Asymmetric Encryption

4.5 Bootstrapping

Bootstrapping is the process of putting all information that a device needs to initiate communication with another component, such as IP addresses and authentication data, onto the device. Figure 7 shows an exemplary result of applying the *Bootstrapping* step of the method. It is explained in more detail by the application example at the end of this section.

Bootstrapping Information - What information do you need to make a device operational? For every device and component in your system, you need to know what information it requires to be able to communicate. List all required pieces of information for every *ObjectFlow*, such as IP addresses, DNS, ports, IDs, usernames, passwords, certificates, cryptographic keys, API Tokens, etc. Also, think about where you would get that information from. Which people, departments, or vendors do you have to contact? Are there paper work, regulations, lead times or deadlines involved?

Bootstrapping Process - How do you deliver bootstrapping information? Once you have the *bootstrapping information*, you have to get it onto your devices and components. If you have only a small number of devices, you can set them up manually during deployment with Medium-Based Bootstrap [26]. If a larger number of devices are involved, it may be better to do a Factory-Bootstrap [26] during production. Both methods can be combined with Remote Bootstrap [26] to allow you to change the *bootstrapping information* dynamically. Annotate each device with a suitable bootstrapping pattern. For other components, delivering the *bootstrapping information* usually boils down to specifying it in a configuration file or as environment variables during deployment.

Registration Process - How do you connect a device to your backend? After you set up all devices with the information they require, you have to register them in your system. For a small number of devices, Manual User-Driven Registration [26] is doable and easy to implement. For larger numbers of devices, Automatic Client-Driven Registration [26] or Automatic Server-Driven Registration [26] are more scalable. Annotate each device with a suitable registration pattern.

Information Storage - How do you store information about known devices? By registering a device in your system, you are adding some general information about the device, a so called *device model*, which contains attributes such as type, name, vendor, firmware version, capabilities, etc. This information is useful for managing your devices and for other components to interact with your devices. A device can provide this information itself with a Device-Driven Model [26]. It could also just chose one Pre-Defined Device-Driven Model [26] from a selection offered by the backend. If a device does not support one of these modes, a Server-Driven Model [26] can be assigned to it by the backend. All *device models* can be stored in a Device Registry [26], where they can be managed centrally and are accessible to other components, even if devices are offline. Add a *StructuredActivityNode* for a Device Registry [26] if necessary and specify all the *ObjectFlows* to and from it where required. Think about if you can combine the Device Registry [26] with another component, for example a Device Gateway [25].

Secure Bootstrapping - How will you secure bootstrapping? Consider Section 4.7 in regards to securing bootstrapping. Make sure that the previous decisions support the required security measurements and adjust if necessary.

Application Example

To get all components up and running, Patrick now has to specify how the initial *Bootstrapping* of communication settings should happen. But first, he lists all the required bootstrapping information, like an API key for the *Sense Sunset* functionality,

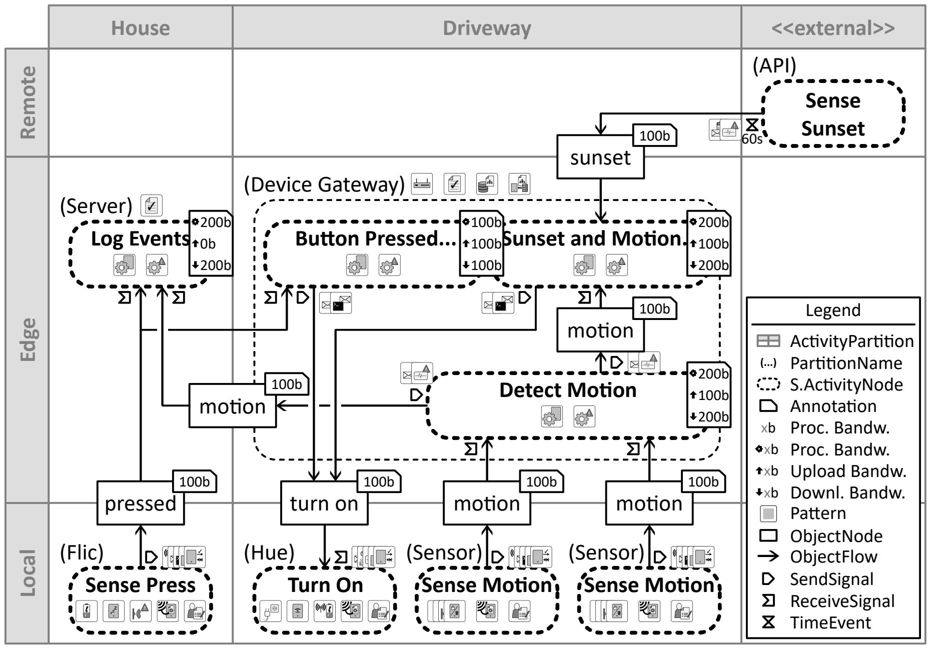

Fig. 7. Exemplary IoT system architecture after *Bootstrapping* step.

or an IP address for the Device Gateway. For the devices, he decides to use Remote Bootstrapping for the initial configuration. As there are only a few devices in the system, he chooses Manual User-Driven Registration to keep things simple. The information should be stored as Server-Driven Models in a Device Registry on the Device Gateway. To secure bootstrapping, Patrick wants to use Digital Signature with Hashing [13]. The result of this step is shown in Figure 7. The following table lists all patterns that have been applied in this step.

Entity	Patterns
Device Gateway *Partition Name*	Device Registry, Server-Driven Model
Sense Press *StructuredActivityNode*	Remote Bootstrap, Manual User-Driven Registration
Turn On *StructuredActivityNode*	Remote Bootstrap, Manual User-Driven Registration
Sense Motion *StructuredActivityNode*	Remote Bootstrap, Manual User-Driven Registration

4.6 Management

At this stage, the IoT system is functional. But to be able to ensure a properly working system well into the future, it will be necessary to manage the system components, for example, to update software. Figure 8 shows an exemplary result of applying the *Management* step of the method. It is explained in more detail by the application example at the end of this section.

Questions

Managing Configuration - How will you change device configuration? From time to time it might be necessary to change device configurations to accommodate changes in the system. This might include switching on or off features, changing communication partners, restarting the device, etc. You can use Remote Device Management [29] to manage device configuration remotely and for large numbers of devices. If so, annotate each device that should be remotely managed with the Remote Device Management [29] pattern and add a new *StructuredActivityNode* for the Remote Device Management server.

Managing Updates - How will you update software on devices? It might also be necessary to update software on devices that are already deployed, for example to introduce new features, fix problems, or close security holes. To avoid having to do this manually, implement Over-the-Air Updates that allow you to push new software and firmware onto devices remotely. Annotate each device for which you want to use Over-the-Air Updates with the pattern.

Management Organization - How will you organize devices? If you have a large number of devices, organizing them can help you manage them more efficiently. Use Organizational Containers to group devices by common attributes, such as type, vendor, location, firmware version, etc. This allows you to introduce filters or automation based on these attributes into other areas, such as Remote Device Management [29], Over-the-Air Updates, or security (see Section 4.7) for more fine-grained control. Add the Organizational Container pattern the the Remote Device Management server.

Management Scalability - How will you handle large numbers of devices? Managing a large number of devices can create its own scalability problems. Batch Campaigns allow you to incrementally roll out configuration and software changes with increased reliability and decreased resource requirements. Add the Batch Campaign pattern to the Remote Device Management [29] pattern where necessary.

Secure Management - How will you secure management? Consider Section 4.7 in regards to securing management. Make sure that the previous decisions support the required security measurements and adjust if necessary.

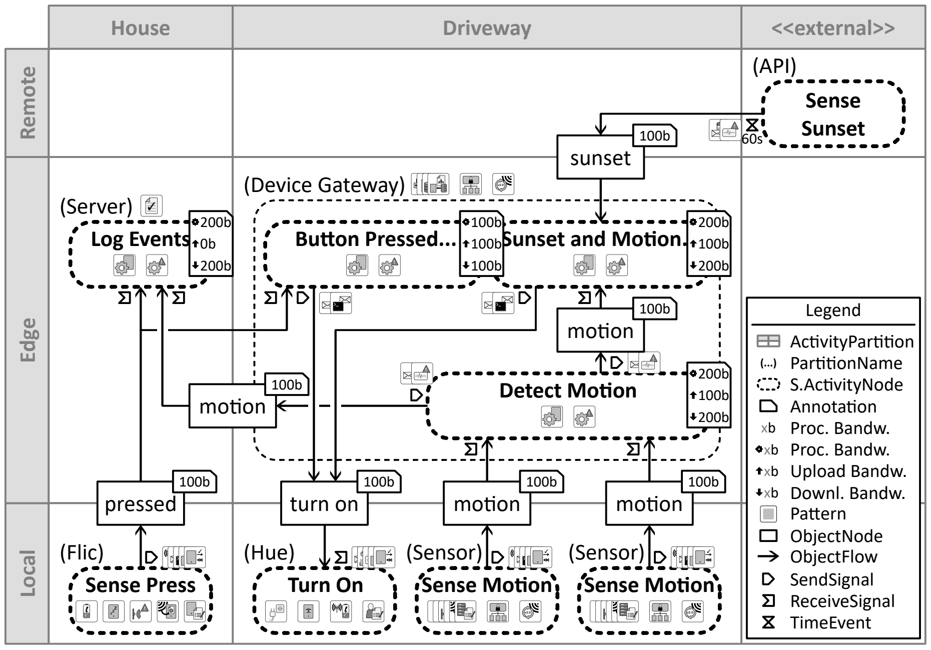

Fig. 8. Exemplary IoT system architecture after *Management* step.

Application Example

At this step, Patrick looks at *Management*. The *Flic* button and the *Philips Hue* light bulb do not support remote management, but for the two motion sensors and the Device Gateway, Patrick wants to implement Remote Device Management. He also wants to use Over-the-Air Updates for these three devices. So he annotates the devices with these two patterns. He decides to skip the management organization and scalability step, as there are only a few devices in the system and that is not about to change soon. To secure management, Patrick wants to use Digital Signature with Hashing [13]. The result of this step is shown in Figure 8. The following table lists all patterns that have been applied in this step.

Entity	Patterns
Device Gateway *Partition Name*	Remote Device Management, Over-the-Air Updates, Digital Signature with Hashing
Sense Motion *StructuredActivityNode*	Remote Device Management, Over-the-Air Updates, Digital Signature with Hashing

4.7 Security

The overall system and all its components should be secured against possible attacks. This includes not only software and network attacks, but also physical attacks against exposed devices. The questions below should already have been answered during the previous steps of the method. You can check them here again to see if you missed something.

Questions

Secure Devices - How will you secure devices? Think about where your devices will be located and how accessible they will be to potential malicious attackers. If they are located somewhere where they could be stolen, a Remote Lock and Wipe [25] mechanism can help you prevent an attacker from accessing data or misusing the device. Add the Remote Lock and Wipe [25] pattern to devices if necessary.

Secure Processing - How will you secure processing? Use Authoritative Source of Data [31] to ensure that the data you process is clean and accurate. Fail Securely [31] when an error occurs so that the failure situation does not create more problems. Add these patterns to processing functionality if necessary.

Secure Communication - How will you secure communication? Communication is one major attack vector for any IoT system and has to be secured properly to avoid data theft or tempering and other attacks on the system. Configuring each device to only accept communication from Trusted Communication Partners [24] that you know. You can go a step further by limiting connection attempts to Outbound-Only Connections [24] from devices. You can also use Whitelists [24] and Blacklists [24] to control communication. Other patterns from existing literature, like Asymmetric Encryption [13], are also applicable here (see additional resources). Annotate devices and other components with these patterns where appropriate.

Secure Bootstrapping - How will you secure the bootstrapping? In addition to securing the communication during bootstrapping and registration as described above, you can use Digital Signature with Hashing [13] to prove the origin and detect tempering of bootstrapping and registration data. The patterns for *secure processing* might also be helpful here. Add these patterns to bootstrapped components where required.

Secure Management - How will you secure management? In addition to securing the communication during management as described above, you can use Digital Signature with Hashing [13] to prove the origin and detect tempering of management data. The patterns for *secure processing* might also be helpful here. Add these patterns to managed components where required.

Additional Resources

In addition to the security patterns mentioned above, there are more security patterns that can be useful described in [13, 19, 31, 32, 33, 36].

Table 1. All patterns that have been applied in the application example.

Entity	Patterns
Sense Press *StructuredActivityNode*	Lifetime Energy-Limited Device, Normally-Sleeping Device, Event-Based Sensing, Remote Bootstrap, Manual User-Driven Registration
Turn On *StructuredActivityNode*	Mains-Powered Device, Always-On Device, Remote Bootstrap, Manual User-Driven Registration
Sense Motion *StructuredActivityNode*	Period Energy-Limited Device, Normally-Sleeping Device, Event-Based Sensing, Remote Lock and Wipe, Remote Bootstrap, Manual User-Driven Registration, Remote Device Management, Over-the-Air Updates, Digital Signature with Hashing
Log Events *StructuredActivityNode*	Local Processing, Event-Based Processing
Button Pressed... *StructuredActivityNode*	Local Processing, Event-Based Processing
Sunset and Motion... *StructuredActivityNode*	Local Processing, Event-Based Processing
Detect Motion *StructuredActivityNode*	Local Processing, Event-Based Processing
Server *PartitionName*	Whitelist
Device Gateway *PartitionName*	Device Gateway, Whitelist, Device Registry, Remote Device Management, Over-the-Air Updates, Digital Signature with Hashing
sunset *ObjectFlow*	Schedule-Triggered Communication, Fire-and-Forget Telemetry, Asymmetric Encryption
motion from **Detect Motion** *ObjectFlow*	Event-Triggered Communication, Reliable Telecommand, Asymmetric Encryption
pressed *ObjectFlow*	Low-Power Short-Range Communication, Event-Triggered Communication, Reliable Telecommand, Trusted Communication Partner, Outbound-Only Connection, Asymmetric Encryption
turn on *ObjectFlow*	Low-Power Short-Range Communication, Event-Triggered Communication, Reliable Telecommand, Pre-Subscription Notification, Trusted Communication Partner, Outbound-Only Connection, Asymmetric Encryption
motion from **Sense Motion** *ObjectFlow*	Low-Power Short-Range Communication, Event-Triggered Communication, Reliable Telecommand, Trusted Communication Partner, Outbound-Only Connection, Asymmetric Encryption

Application Example

Patrick has already considered security in parallel to the previous steps, as depicted by Figure 2. Thus, after checking for a final time, he decides that there are no additional security measurements required at this point. Patrick is now finished with the method. The end result is the high level pattern-annotated architecture diagram shown in Figure 8. Table 1 lists all patterns that have been applied during the method.

Patrick can now uses this diagram to look for available solutions for the missing parts in the system. The details and patterns specified in the diagram help him find motion sensors and a Device Gateway that support the required functionality. He decides to implement the server component himself. With this plan and the diagram, he can now talk to the other involved stakeholders, in this case his client. Once the plan is approved, he uses the pattern-annotated diagram as a basis for further specifying the details of the server component, as well as the three functionality nodes that he has to implement on the *Device Gateway*.

5 Conclusion and Future Work

In this paper, we showed our pattern-based method for designing IoT systems, which can be used by practitioners to build up an UML-based and pattern-annotated architecture diagram. This diagram can be used as basis for further architecture refinements, as guide for finding and selecting existing solutions, and to communicate with other stakeholders.

We see a few areas, where future work could improve this method. In its current state, there are some steps in our method that are not backed by patterns. Here, it could be interesting to investigate, if patterns in those areas could be found. We also had little space to describe additional resources and how they can be applied in detail. Future work could improve upon this and provide a more detail explanation and integration of additional resources. Another area for future work is tooling support for this method, which we think could provide value to practitioners. It could reduce the amount of manual work required to execute the method and could be integrated with other software tooling for easier use. It could also be interesting to explore existing approaches for linking patterns to concrete implementations [5, 6] and to technology specific patterns [7] to ease the pattern application process for the practitioner. Finally, we plan to evaluate this method in real world projects, which could give us an idea of its suitability and of possible improvements.

Bibliography

[1] Secpatt (2019), `https://www.secpatt.at/patterns/`

[2] Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York (1977)

[3] Bayuk, J.L., Horowitz, B.M.: An architectural systems engineering methodology for addressing cyber security. Systems Engineering **14**(3), 294–304 (2011)

[4] Eloranta, V.P., Koskinen, J., Leppänen, M., Reijonen, V.: Designing distributed control systems: A pattern language approach. Wiley series in software design patterns, Wiley, Hoboken, NJ (2014)

[5] Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F.: Efficient pattern application: Validating the concept of solution implementations in different domains. International Journal on Advances in Software **7**(3&4), 710–726 (2014)

[6] Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F.: From pattern languages to solution implementations. In: Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014). pp. 12–21. IARIA, Wilmington, DE (2014)

[7] Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F., Hadjakos, A., Hentschel, F., Schulze, H.: Leveraging pattern application via pattern refinement. In: Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC) (2016)

[8] Falkenthal, M., Breitenbücher, U., Leymann, F.: The nature of pattern languages. In: Pursuit of Pattern Languages for Societal Change. pp. 130–150. tredition (2018)

[9] Fehling, C.: Cloud Computing Patterns: Identification, Design, and Application. Dissertation, University of Stuttgart, Stuttgart (2015)

[10] Fehling, C., Leymann, F., Retter, R.: Your coffee shop uses cloud computing. IEEE Internet Computing **18**(5), 52–59 (2014). https://doi.org/10.1109/MIC.2014.101

[11] Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications. Springer, Wien (2014). https://doi.org/10.1007/978-3-7091-1568-8

[12] Feldhusen, J., Bungert, F.: Pattern languages to create a holistic methodology for product development and to derive enterprise-specific engineering guidelines. In: Industrial Engineering and Ergonomics, pp. 131–141. Springer (2009)

[13] Fernandez, E.B.: Security Patterns in Practice: Designing Secure Architectures Using Software Patterns. Wiley (2013)

[14] Harrison, N.B., Avgeriou, P., Zdun, U.: Using patterns to capture architectural decisions. IEEE software **24**(4), 38–45 (2007)

[15] Hehenberger, P., Vogel-Heuser, B., Bradley, D., Eynard, B., Tomiyama, T., Achiche, S.: Design, modelling, simulation and integration of cyber physical systems: Methods and applications. Computers in Industry **82**, 273–289 (2016)

[16] Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, Boston, Massachusetts (2004)

[17] ISO: Iso/iec 25010:2011 (2011), `https://www.iso.org/standard/35733.html`

[18] Khaitan, S.K., McCalley, J.D.: Design techniques and applications of cyberphysical systems: A survey. IEEE Systems Journal **9**(2), 350–365 (2014)

[19] Kienzle, D.M., Elder, M.C., Tyree, D., Edwards-Hewitt, J.: Security patterns repository version 1.0 (2002), `http://www.scrypt.net/~celer/securitypatterns/repository.pdf`

[20] Miorandi, D., Sicari, S., Pellegrini, F.d., Chlamtac, I.: Internet of things: Vision, applications and research challenges. Ad hoc networks **10**(7), 1497–1516 (2012)

[21] Object Management Group: Omg unified modeling language (omg uml) (05122017), `https://www.omg.org/spec/UML/2.5.1/PDF`

[22] Petroulakis, N.E., Spanoudakis, G., Askoxylakis, I.G., Miaoudakis, A., Traganitis, A.: A pattern-based approach for designing reliable cyber-physical systems. In: 2015 IEEE Global Communications Conference (GLOBECOM). pp. 1–6 (2015)

[23] Porter, R., Coplien, J.O., Winn, T.: Sequences as a basis for pattern language composition (2005)

[24] Reinfurt, L., Breitenbücher, U., Falkenthal, M., Fremantle, P., Leymann, F.: Internet of things security patterns. In: Proceedings of the 24th Conference on Pattern Languages of Programs (PLoP) (2017)

[25] Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of things patterns. In: Proceedings of the 21st European Conference on Pattern Languages of Programs (EuroPLoP). ACM (2016)

[26] Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of things patterns for device bootstrapping and registration. In: Proceedings of the 22nd European Conference on Pattern Languages of Programs (EuroPLoP). EuroPLoP '17, ACM, New York, NY, USA (2017)

[27] Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of things patterns for devices. In: Proceedings of the Ninth International Conferences on Pervasive Patterns and Applications (PATTERNS) 2017. pp. 117–126. Xpert Publishing Services (2017)

[28] Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of things patterns for devices: Powering, operating, and sensing. International Journal on Advances in Internet Technology **10**(3 & 4), 106–123 (2017)

[29] Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of things patterns for communication and management. LNCS Transactions on Pattern Languages of Programming (2019)

[30] Reinfurt, L., Falkenthal, M., Leymann, F.: Where to begin - on pattern language entry points. SICS Software-Intensive Cyber-Physical Systems **35**(in press) (2020)

[31] Romanosky, S.: Security design patterns (2001), `http://www.cgisecurity.com/lib/securityDesignPatterns.html`

[32] Schumacher, M.: Firewall patterns. In: Proceedings of the 8th European Conference on Pattern Languages of Programms (EuroPLoP '2003) (2003)

[33] Schumacher, M., Fernandez, E.B., Hybertson, D., Buschmann, F., Sommerlad, P.: Security Patterns: Integrating Security and Systems Engineering. Wiley (2005)

[34] Tidwell, J.: Designing interfaces: Patterns for effective interaction design. O'Reilly Media, Inc (2010)

[35] Uzunov, A.V., Fernandez, E.B., Falkner, K.: Engineering security into distributed systems: A survey of methodologies. J. UCS **18**(20), 2920–3006 (2012)

[36] Yoder, J., Barcalow, J.: Architectural patterns for enabling application security. `https://www.idi.ntnu.no/emner/tdt4237/2007/yoder.pdf`

[37] Zimmermann, O., Zdun, U., Gschwind, T., et al.: Combining pattern languages and reusable architectural decision models into a comprehensive and comprehensible design method. In: Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008). pp. 157–166 (2008)

All links were last accessed on 08.08.2019

Impact of Application Load in Function as a Service

Johannes Manner and Guido Wirtz

DSG, University Bamberg, An der Weberei 5, 96047 Bamberg, Germany
{`johannes.manner,guido.wirtz`}`@uni-bamberg.de`

Abstract. Function as a Service (FaaS) introduces a different notion of scaling than related paradigms. The unlimited upscaling and the property of downscaling to zero running containers leads to a situation where the application load influences the number of running containers directly. We propose a combined simulation and benchmarking process for cloud functions to provide information on the performance and cost aspect for developers in an early development stage. Our focus in this paper is on simulating the concurrently running containers on a FaaS platform based on different function configurations. The experiment performed serves as a proof of concept work and emphasizes the importance for design decisions and system requirements. Especially for self-hosted FaaS platforms or resources bound to cloud functions like database connections, this information is crucial for deployment and maintenance.

Keywords: Serverless Computing, Function as a Service, FaaS, Benchmarking, Simulation, Load Profile

1 Introduction

As in every virtualization technology, a cloud function container faces some performance challenges when it is created and executed for the first time. To estimate the quality of serivce a system delivers, benchmarking applications is crucial. Huppler [1] stated that a benchmark should be *relevant*, *fair*, *verifiable*, *economical* and *repeatable*. There does exist a bunch of experiments, e.g. [2,3,4,5,6], executed in the FaaS domain to assess this new paradigm in the cloud stack.

We investigate these benchmarks based on the requirements of Huppler. All of these benchmarks make a performance evaluation of one or more FaaS platforms compared to each other or related technologies like Virtual Machine (VM) based solutions. The biggest issue in general is the *repeatability* of a benchmark since the targeted field is highly evolving. Another problem is the lack of information about the settings and other influential factors of the mentioned experiment. Results are discussed in detail for every of these publications, but only a few of them describe all the necessary steps to repeat the experiment and verify the findings, as Kuhlenkamp and Werner [7] ascertained. All the benchmarks in their literature study are FaaS related and conducted since 2015. They gave each experiment a score between 0 and 4 to assess quality of the presented work. Workload generator, function implementation, platform configuration and other used services are the categories of their systematic literature study. The mean average was 2.6, which indicates that a lot of information is missing in the conducted benchmarks. Only 3 out of 26 experiments supplied all preconditions and parameters needed to make it possible to reproduce the presented results. Therefore, results of different benchmarks are often not comparable to each other. The first category, generation of load patterns and their topology, is the least discussed item. Authors of FaaS benchmarking papers only write in every third publication about the load pattern aspect.

As the load pattern topology has a major influence on the scaling behavior of a cloud function platform, we focus on this aspect here. This is also important for software architects constructing hybrid architectures which need information about the incoming request rate in the non-FaaS part of their systems. Otherwise, the FaaS part of applications can cause Distributed Denial of Service (DDos) attack on other parts. Our paper stresses this aspect in particular by (i) discussing different ways to specify load patterns, (ii) proposing a workflow for a combined FaaS simulation and benchmarking process and (iii) presenting a methodology to compute the number of running instances out of the respective load trace.

The outline of the paper is as follows. Section 2 discusses related work and answers the first contribution, which load generation tools are suited to specify application workloads. Section 3 proposes a generic workflow for a simulation and benchmarking process of cloud functions and picks a single aspect, the number of concurrently running functions as a proof of concept. The paper concludes with a discussion in Section 4 and an outlook in Section 5.

2 Related Work

2.1 Benchmarking FaaS

The open challenges Iosup and others [8] mentioned in their publication for Infrastructure as a Service (IaaS) benchmarking are partly-open challenges for FaaS as well. There is currently a lack of methodological approaches to benchmark cloud functions consistently. Malawski and others [6,9] conducted scientific workflow benchmarks and built their benchmarking pipeline based on the *serverless framework* [1]. They publish their benchmarking results continuously, but do not include simulations, which would reduce cost and time. Similar to this approach, Scheuner and Leitner [10] introduced a system, where micro and application benchmarks are combined. Especially the micro benchmarking aspect is interesting for a consistent FaaS methodology since typically a single cloud function is the starting point. Three different load patterns are part of

[1] `https://serverless.com/framework/`

their contribution but hidden in the implementation of their system and therefore not directly mentioned, as in many other FaaS publications. These initial benchmarks focusing on a single aspect in isolation are important steps to understand the impact on system design and execution, but they are quite difficult to setup and need a lot of time for execution, as Iosup already mentioned for IaaS benchmarking. So, [8] proposes a combination of simulating small sized artificial workloads and conducting real world experiments as the most promising approach to get stable results with least effort in time and money.

2.2 Load Patterns in Conducted Experiments

The "job arrival pattern" [8] is critical for the performance of any System Under Test (SUT). Especially in FaaS, where scaling is determined by the given input workload. To perform repeatable benchmarks and enable a simulation of cloud functions under different external circumstances, the documentation of load patterns is critical. As mentioned in the introduction, this is the least discussed aspect, but some authors explained their workload in detail, as discussed by [7]. These descriptions are not sufficient:

- McGrath and Brenner [11] performed a concurrency test and a backoff test. The concurrency test featured 15 test executions. Each of them at 10 seconds intervals with an increasing number of concurrent request. For the first test execution only 1 request was started and in the last execution 15 concurrent requests were submitted. This was repeated 10 times. The backoff test performed a single invocation from 1 to 30 minutes pausing time between the invocations to investigate the expiration time of a cloud function container and the impact of cold start on execution performance.
- Lee and others [4] focused on concurrency tests. First they measured the function throughput per second by invoking the cloud functions 500, 1,000, 2,000, 3,000 and 10,000 times. The time between invocations was not mentioned. Therefore, it is not clear if the second call used the already warm containers from the first execution. Furthermore, they investigated different aspects with 1 request at a time and 100 concurrent requests and a few other settings, but also not informed the reader about wait time between calls or the exact distribution.
- Figiela and others [12] conducted two CPU intensive benchmarks. The first one was executed every 5 minutes and invoked the different functions once. The second experiment used a fork-join model and executed the tasks in parallel for 200, 400 and 800 concurrent tasks. The number of repetitions and the corresponding wait time between them were not mentioned and maybe not present.
- Back and Andrikopoulos [2] used *fast fourier transformation*, *matrix manipulation* and *sleep* use cases for their benchmark. They parameterized each function and executed each combination once a day on three consecutive days. It is unclear to the reader, if all of these measurements resulted in a cold start on the respective providers. Also the results are prone to outliers since the sample size with 3 executions per combination can distort findings.

– Das and others [13] implemented a sequential benchmark of cloud and edge resources, where the time of two consecutive invocations was between 10 and 15 seconds to avoid concurrent request executions. There is no information how the authors dealt with the first invocation of a cloud function.

Manner and others [14] focused on the cold start overhead in FaaS. Therefore, they defined a sequential load pattern to generate pairs of a single cold and warm start to compare the performance on a container basis. Warm starts were executed 1 minute after the cold execution returns. After the pair was executed, the pattern paused for 29 minutes to achieve a shutdown of the container. W.r.t. the load pattern aspect, the experiments in this publication are reproducible and all necessary information is described to repeat them.

All the presented workloads are artificial load patterns, where some reductions are made for simplicity and to assess a single detail or use case in FaaS. It is often unclear if the authors used an established load generation tool or implemented a proprietary interface for submitting the workload. There is currently a lack in experiments, which use real world load traces.

2.3 Load Generation Tools

Before discussing load generation tools, the kind of application load is important for any benchmark or simulation. Schroeder and others [15] defined three of them: Closed, open and partly-open systems. Closed systems can predict, based on other parts of the system, how many incoming request will arise. In contrast, the workload of an open system is not predictable since users access the service randomly via an interface. Partly-open is a combination of both.

We only focus on open systems since a single cloud function is in focus of our work and has therefore no other dependencies. There exists a recent study about *workload generators for web-based systems* [16], where a comprehensive collection is presented and a lot of generation tools are compared. For benchmarking FaaS, an arrival rate of requests is the needed input. Therefore, we picked two tools to generate workloads as a reference here. Tools like *JMeter* [2] focus on controlled workloads with constant, linear or stepwise increasing loads. This behavior is especially important to generate clean and clear experimental setups to isolate different aspects under investigation. Based on these ideas, we also implemented some benchmarking modes in our prototype[3] to control the execution of requests based on our needs and added some instrumentation to compare the execution time on the platform and on a local machine, submitting the requests. On the other hand, there are tools to model real world load traces based on seasonal, bursty, noisy and trend parts like LIMBO [17]. LIMBO enables the generation of a load pattern based on an existing trace or via combination of mathematical functions. In contrast to JMeter, where the load can be directly submitted, LIMBO decouples the load generation and the submission via another tool, as suggested by [16].

[2] `http://jmeter.apache.org/`
[3] `https://github.com/johannes-manner/SeMoDe`

3 FaaS Benchmarking and Simulation

3.1 Combined Workflow

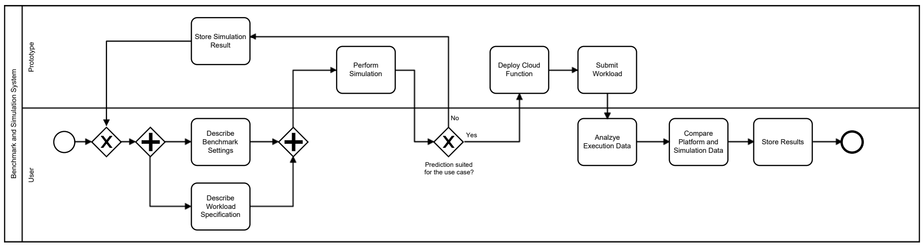
Benchmark and Simulation System
Prototype
User
Store Simulation Result
Perform Simulation
Deploy Cloud Function
Submit Workload
No
Yes
Describe Benchmark Settings
Describe Workload Specification
Prediction suited for the use case?
Analzye Execution Data
Compare Platform and Simulation Data
Store Results

Fig. 1: Generic Pipeline for FaaS Benchmarking

Figure 1 presents a generic pipeline for FaaS benchmarking inspired by Iosup [8]. The SUT is not explicitly mentioned since a single cloud function is the SUT in our approach. Memory setting, the size of the deployment artifact etc. [14] directly influence the execution time and are therefore relevant, in combination with the load pattern, to assess the concurrently running containers.

After providing the cloud function, the load pattern and the mentioned metadata, our prototype starts the simulation. After simulation is done, the user has to decide, if the simulated values are suited for the use case, e.g. if the number of concurrently running instances not exceeding a limit, or, if he has to adjust the values and starts another simulation, e.g. raising the memory setting and reducing the overall execution time. In the latter case, the prototype stores this interim result for a later comparison with the next simulation round, where a developer can assess which setting results in better cost and performance.

If the simulation is satisfying for the user, our prototype deploys the function using *serverless framework* and submits the workload based on the load pattern. Our prototype uses synchronous Representational State Transfer (REST) calls to generate events on the FaaS platform as introduced in [14]. This behavior is similar to the *direct executor model* as proposed by [9]. Subsequently, the user analyses platform execution data and compares the results with predicted values of the simulation. Finally, the results are stored for further improving the simulation framework, proposed in a prior paper [18].

3.2 Simulating Number of Cloud Function Containers

This section focuses on a single piece of the presented pipeline in Figure 1: *Perform Simulation*. We investigate only one aspect of this piece: *Number of running containers*. An important aspect is execution time w.r.t. to different function configurations. Also the input of functions highly influences the runtime performance, e.g. sorting algorithms. We tackle this problem of varying execution times in future work, when refining the simulation engine. Further aspects are the associated cost impact, effects on used backend services like cloud databases etc. If the simulation exposes a high number of concurrently running containers and the concurrency level is problematic, the developer could throttle the incoming requests by using a queue etc.

Algorithm 1 is implemented by our prototype. A comparison to used scheduling algorithms in open source FaaS platforms is outstanding. Currently, the simulation uses the mean average execution time (*exec*) of the investigated cloud function, mean cold start time (*cold*) and idle time for container shutdown (*shutdown*). The *timeStamps* are a list of double values marking the start time of a request and are created manually. Statistical deviations are not included in this proposed simulation approach, but planned for future work. The gateway spawns events and triggers the function under test. Multi-tenancy is not included in our algorithm, but has an impact on performance and execution time as HELLERSTEIN and others [19] stated.

Algorithm 1 Basic Simulation - Number of Containers

```
 1: procedure SIMULATE($exec, cold, shutdown, timeStamps$))
 2:     for $time$ in $timeStamps$ do
 3:         $checkFinishedContainers(time)$
 4:         $shutdownIdleContainers(time, shutdown)$
 5:         if $idleContainerAvailable()$ then
 6:             $pickIdleAndExecute(exec)$
 7:         else
 8:             $spinUpAndExecute(cold, exec)$
 9:         end if
10:     end for
11:     $shutdownAllContainers()$
12:     $generateContainerDistribution()$
13: end procedure
```

In line 3, the program checks, if some of the containers have finished their execution at *time* and sets these containers in an idle state. The next function shuts all idle containers down, which exceed the shutdown time. At this point, the internal state of the simulation is clean and the next request can be executed either from an already warm container (line 5, 6) or a new instance (line 8), which is affected by a cold start. If all request are served, the prototype produces a distribution, how many containers are running on the basis of seconds.

4 Discussion

Figure 2 depicts an initial load trace and two corresponding simulations. The colored numbers are counts on a second basis and show the number of incoming requests (orange) and the number of concurrently running containers (yellow and gray). The input trace is artificially created and the values [4] for these two simulations are chosen w.r.t. a prior investigation [14] [5].

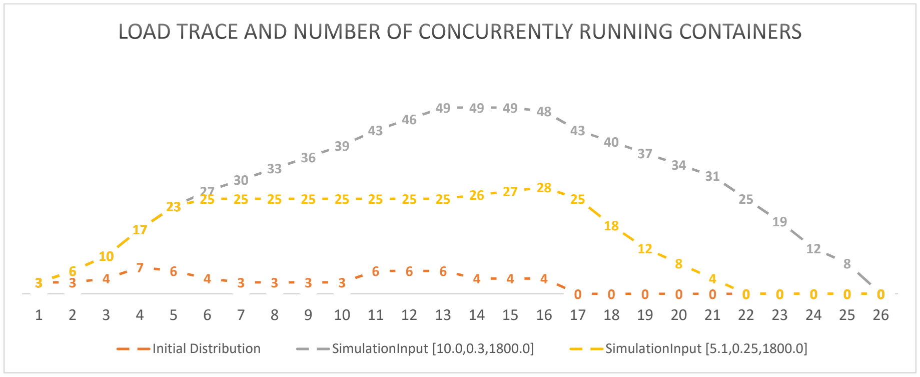

Fig. 2: Example Distribution for an Artificial Load Trace

The execution time of the gray run (5.1) is roughly twice the execution time of the yellow run (10.0). As for many FaaS platforms, like AWS Lambda [6] and Google Cloud Functions [7], the CPU resources are directly coupled with the memory setting. We suppose, that for example the yellow cloud function is deployed with a memory limit of 256 MB RAM, whereas the gray cloud function is restricted to 128 MB RAM. Assumed, that the two functions are implemented in Java, the cold start time is not affected in the same way since the JVM startup is resource intensive in both cases. The shutdown time (1800.0) has no effect in this example since the considered interval is too short.

Gray and yellow graphs show a start-up, an execution and a tear-down phase. Our artificial distribution simulates a moderate load with a few invocations per second. The start-up phase is similar for the first 5 seconds since after 5 seconds the first containers are reused in the yellow simulation. Our execution phase is only five seconds for the gray (second 11 to 16) compared to ten seconds for the yellow simulation (second 6 to 16), but shows the impact of the supposed runtime configuration on the number of running containers. For self-hosted FaaS platforms or resources bound to cloud functions like database connections, the difference between 28 or 49 concurrently running containers influence system requirements and design decisions. The tear-down in the yellow case happens faster due to the shorter execution time. The output load trace is missing in this simulation.

[4] Compare the input values to Algorithm 1 - SimulationInput[exec,cold,shutdown].
[5] Source code, parameters and input trace are available on GitHub: `https://github.com/johannes-manner/SeMoDe/releases/tag/summersoc13`
[6] `https://aws.amazon.com/lambda`
[7] `https://cloud.google.com/functions/`

5 Future Work

The aim is to implement the suggested simulation and benchmarking pipeline in our prototype. Therefore, the next step is to include an automated data picking facility as MALAWSKI and others [9] already implemented.

Our simulation model is based on a few parameters without statistical deviation to keep the system deterministic. We want to extend the simulation in this directions and also include the output load pattern of our simulation since this output is maybe the input for another component of the overall (hybrid) application. Furthermore, we want to conduct a few benchmarks on constant, linear and bursty workloads to refine our simulation model and perform a realistic proof-of-concept of our work and include the multi tenancy aspect.

To conclude, the topology of load pattern has a major influence on the number of running containers on the FaaS platforms. Our paper stresses this aspect in particular and puts emphasis on the lack of documentation in conducted experiments from the literature. The presented simulation is a first step in our overall simulation approach towards predictability of platform behavior.

References

1. K. Huppler. The art of building a good benchmark. In Raghunath Nambiar and Meikel Poess, editors, *Performance Evaluation and Benchmarking*, pages 18–30. Springer, 2009.
2. T. Back and V. Andrikopoulos. Using a Microbenchmark to Compare Function as a Service Solutions. In *Service-Oriented and Cloud Computing*. Springer, 2018.
3. D. Jackson and G. Clynch. An investigation of the impact of language runtime on the performance and cost of serverless functions. In *Proc. WoSC*, 2018.
4. H. Lee et al. Evaluation of Production Serverless Computing Environments. In *Proc. WoSC*, 2018.
5. W. Lloyd et al. Improving Application Migration to Serverless Computing Platforms: Latency Mitigation with Keep-Alive Workloads. In *Proc. WoSC*, 2018.
6. M. Malawski et al. Benchmarking Heterogeneous Cloud Functions. In Dora B. Heras and Luc Bougé, editors, *Euro-Par 2017: Parallel Processing Workshops*, pages 415–426. Springer International Publishing, 2018.
7. J. Kuhlenkamp and S. Werner. Benchmarking FaaS Platforms: Call for Community Participation. In *Proc. WoSC*, 2018.
8. A. Iosup et al. IaaS Cloud Benchmarking: Approaches, Challenges, and Experience. In *Proc. MTAGS*, 2012.
9. M. Malawski. Towards Serverless Execution of Scientific Workflows HyperFlow Case Study. In *Proc. WORKS*, 2016.
10. J. Scheuner and P. Leitner. A Cloud Benchmark Suite Combining Micro and Applications Benchmarks. In *Proc. ICPE*, 2018.
11. G. McGrath and P. R. Brenner. Serverless computing: Design, implementation, and performance. In *Proc. ICDCSW*, 2017.
12. K. Figiela et al. Performance evaluation of heterogeneous cloud functions. *Concurrency and Computation: Practice and Experience*, 2018.
13. A. Das et al. EdgeBench: Benchmarking edge computing platforms. In *Proc. WoSC*, 2018.

14. J. Manner et al. Cold Start Influencing Factors in Function as a Service. In *Proc. WoSC*, 2018.
15. B. Schroeder et al. Open Versus Closed: A Cautionary Tale. In *Proc. NSDI*, 2006.
16. M. Curiel and A. Pont. Workload generators for web-based systems: Characteristics, current status, and challenges. *IEEE Communications Surveys & Tutorials*, 20(2):1526–1546, 2018.
17. J. von Kistowski et al. Modeling and extracting load intensity profiles. *ACM Transactions on Autonomous and Adaptive Systems*, 11(4):1–28, 2017.
18. J. Manner. Towards Performance and Cost Simulation in Function as a Service. In *Proc. ZEUS (accepted)*, 2019.
19. J. M. Hellerstein et al. Serverless Computing: One Step Forward, Two Steps Back. In *Proc. CIDR*, 2019.

Coverage criteria for integration testing of serverless applications

Stefan Winzinger and Guido Wirtz

DSG, University Bamberg, An der Weberei 5, 96047 Bamberg, Germany
{stefan.winzinger,guido.wirtz}@uni-bamberg.de

Abstract. Serverless computing is a popular computing model offered by many cloud providers. Because of the statelessness of the functions used, applications can be scaled up dynamically. This is particularly interesting for applications expecting irregularly heavy peak loads. The combination of many serverless functions with other services builds a complex system. By using integration tests, the behavior of the complex system can be tested. However, it is hard to determine the relevant test cases and to check if the current set of test cases is good enough. Therefore, we discuss how approaches checking the adequacy of a test case set can be applied for serverless applications during integration testing and how characteristics of a serverless application can support the assessment of test cases.

Keywords: serverless computing · FaaS · coverage criteria · integration testing · model-driven testing

1 Introduction

After the introduction of Amazon's "AWS Lambda" [1] in 2014, serverless computing became quite popular. Also other big IT companies like Google [2], IBM [3], Microsoft [4] offer serverless computing. But there are also more and more open source projects (like OpenFaaS [5] and OpenLambda [6]) available indicating the demand for serverless computing.

Stateless serverless functions are the basis for serverless applications. The statelessness of the functions enables a dynamic scaling of the application [7] which is particularly interesting for applications with heavy peak loads. Since there are no costs if a function is idle and not used, serverless computing is also interesting for applications with an irregular work load. The scaling of the functions is done by the cloud platform provider who executes small code snippets, usually written in languages like Java, JavaScript, C# or Python, whereas the developer has no control over the resources on which the code runs [8]. Therefore, both costs and performance can be improved by using serverless computing [9], [10].

Serverless applications consists of numerous, short-lived, stateless functions [11] which requires tools for the development of complex serverless applications. Single serverless functions can be tested easily in isolation by using unit tests.

However, the behavior emerging from the interaction of many serverless functions with each other and other services is hard to predict and test in isolation. The potential parallel execution of services makes it even harder. In order to reduce the complexity of the system and apply a systematic testing approach, a model of the system that concentrates on the relevant parts of the system can be helpful.

However, there is not much work available yet which describes the information needed for modeling a serverless application for integration testing.

Chang et al. [12] present a tool using the execution log of a serverless application to visualize its structure and runtime behavior. Using a graph for the analysis of a system is also used in many areas (e.g, [13], [14], [15]) which is not directly applicable to complex serverless applications since they focus on different aspects. General approaches for testing the adequacy of integration test cases are discussed by [16].

In [17], we introduced a model for the representation of serverless applications based on a call graph that presents and evaluates the basic benefits of using such a structure for the representation. This model shows some possibilities for the representation of the specific characteristics of a serverless application. Additionally, in [18] we discussed which approaches and characteristics of a serverless application can be used as coverage criteria.

In this paper, we extend our previous work by discussing how flow-based approaches and serverless characteristics can be applied for testing the adequacy of integration test cases and how this can be supported by a model representing a serverless application.

The paper is structured as follows. We describe flow-based coverage and propose how it can be applied for serverless applications in Section 2. Section 3 demonstrates how certain characteristics of a serverless application can be used for integration testing. An outlook on future work is given in Section 4, whereas Section 5 draws a conclusion.

2 Flow-based adequacy

This section discusses how classical approaches like control flow and data flow can be used for the evaluation of integration test cases of a serverless application. By modeling the control flow or the data flow of the system, test case criteria can be created. This helps verify existing test cases and determine if additional test cases are needed. Additionally, it can support the creation of missing test cases by evaluating the model and help map missing test cases to the system under test.

In the following, we discuss how these techniques can be applied for a serverless application during integration testing and suggest how it can be supported by a model.

2.1 Control flow

A classical testing approach as shown in [19] is the examination of an application where the control flow is considered as a graph. By building blocks of statements being mapped to single nodes and edges representing calls to following statements, a graph can be created. Based on this graph, coverage criteria can be applied requiring the execution of all nodes, all edges or all paths of the graph. If this approach is applied on code level for a serverless application, not only the integration of the different services, but also the structure within a function is tested which has to be known. Thus, code has to be covered which should be tested explicitly in isolation but is not required in the integration. Since these parts are tested additionally, the size of a test case set increases tremendously for a complex system, in particular if certain combinations of nodes have to be tested like for the coverage of all paths.

Therefore, building a graph on this fine-granular level produces a graph which is too complex and focuses not only on the relations between its components but also on the relations within its single components which is the job of unit testing.

We suggest to use a model of the application where only the relations between the resources are considered in order to concentrate on the communication between the resources. The test criteria are adapted and applied for this abstraction level.

The model introduced in [17] represents a serverless application as a call graph and is well-adaptable for the requirements needed. The model focuses on the abstract call hierarchy between its resources and is therefore an ideal mean for the application of control flow coverage criteria. By using the call graph as a basis for the coverage, a model is available which is applicable for an existing system and can easily be adapted.

Inspired by [20] where criteria are defined on module level, we suggest the following coverage criteria for serverless applications which can be applied by using the previous model:

- *All-resources* requires that every resource is executed at least once. A resource is the instance of a service like a serverless function or a data storage.
- *All-resource-relations* requires that every call between resources is executed (e.g., all edges between the nodes are covered).
- *All-resource-sequences* requires that every sequence of resources is executed (e.g., all paths of the graph are called).

By demanding the coverage of all resources, it is guaranteed that each resource was called at least once. This can be used for continuous deployment to see if a instance which was added recently was deployed correctly and fulfills its basic functionality. The size of coverage units depends directly on the deployed resource and thus a complicated instrumentation and adaption of the system is not required.

A coverage of all edges in the graph supports the testing of the communication of resources with their neighbors. Thus, not only the local isolated behavior of the resources is tested but also the integrated behavior of a resource.

By using the model described, all potential workflows can be detected and demanded to be covered. Consequently, it covers not only technical aspects but also the global context of the resource. Often it is hard to calculate all feasible passes or even impossible if e.g., circles are present.

Therefore, the model must contain at least all resources of the application. If all relations have to be covered, all relations have to be modeled too. However, if all sequences of the resources shall be covered, more information is useful in order to prevent the creation of too many infeasible paths. By adding information like the call order of resources, constraints for calls, asynchronous or synchronous invocation types, write or read data access or the number of potential calls of resources, the number of paths can be reduced. As already shown in [17], such a model can be supported by a tool.

2.2 Data flow coverage

The control flow considered in the previous section ignores the data used and passed. By considering the data being used by different resources, a direct influence between the resources can be tested. Usually, data needed by a serverless function are passed by its parameters as input. However, if a serverless application gets more complicated and more data are needed, these data are transferred by using data storages where the values are temporarily stored.

Similar to the control flow of coverage criteria, we suggest to consider the data flow between the resources used in a serverless application. The graph used in the previous section can be used to support the identification of data flows by annotating edges with the data being transmitted.

Similar to the control flow, we only consider data flows where several resources are involved. Thus, inspired by [20], we suggest the following criteria where x is a definition of a value within a serverless function which is used by another resource:

- *All-resource-defs:* requires that every x is at least used once in another resource without being redefined before its usage.
- *All-resource-uses* requires that every x is used by all other usages of x in other resources without being redefined before its usage.

These coverage criteria can be applied for a serverless application and are relevant since a lot of data are transferred within a serverless application because of the statelessness of the serverless functions. Tracking these data flows can be supported by the model representing the system but needs further support.

However, the identification of def-use-pairs is difficult since it requires insights into the code structure of the functions and the transmission of its values. Not only tracking the data over many resources is difficult, but also an analysis might be hard for functions being deployed in a running system since dependencies over many resources might have to be recalculated or tracking data have to be adapted.

3 Adequacy of characteristics

This section shows how characteristics of a serverless application can be used to check the adequacy of test cases and how this can be supported by the model.

3.1 Parallelism

Having the possibility to run serverless functions in parallel is a characteristic of a serverless application. Parallelism can cause errors which have to be detected. The parallelism is enabled by the scalability of the serverless functions. But also a parallel access of the same application can cause parallel workflows being responsible for errors.

If at least two workflows access the same data in parallel where at least one workflow writes data, race conditions may occur. The detection of these hot spots can be supported on integration level by modelling data storages with resources accessing them by either writing or reading data.

In order to test the criticality of the hot spots, we suggest to test read and write accesses to the data storage in all possible orders. Additionally, if the workflow is modelled like suggested in the previous section, the accesses to the data base should be tested in different contexts for all possible orders.

However, testing the hot spots for all possible orders not only requires an identification of all these situations but also a framework which enables the execution in different orders.

3.2 Execution time

There are several factors influencing the execution time of a serverless application which can be considered for testing. The cold start time needed for the containerization of a serverless function if this function is not already loaded [21] is characteristic for serverless applications. Therefore, some executions of serverless functions take longer.

But the execution time is also influenced by the infrastructure assigned to a serverless function. Even if there are not many resources assigned to a serverless function, the platform provider can decide to load the container of the serverless function to a faster machine [22]. This results in unpredictable execution times. Therefore, if the application is tested on a faster or slower platform than is actually used in production, potential failures can be obfuscated. These failures are either time out errors resulting from a longer execution time or race conditions. The ordering of the workflow might be disturbed by different execution times. Thus, race conditions can occur where data are used by several workflows where at least one writes data.

In order to cover the application, profiles for the functions have to be created giving information about the distribution of the execution times of the serverless functions.

Therefore, we suggest the following coverage criteria for testing a serverless application.

- *All-min-time* requires that every serverless function has to be executed with its minimum time at least once.
- *All-max-time* requires that every serverless function has to be executed with its maximum time at least once.

– *All-max-min-time* requires that every serverless function has to be executed with its minimum and maximum time at least once.

In order to apply the criteria, a dynamic measurement of the time behavior of the functions is necessary after the deployment. Additionally, time profiles are needed in order to categorize the times measured.

Since the different execution times might also be influenced by the contexts in which the functions were triggered, the functions can also be tested on a finer level by measuring the time profile for a certain context (e.g., a certain workflow).

3.3 Access rights

Security is a relevant aspect in a serverless application. Access rights are assigned to the serverless functions regulating that only resources are accessed and used which are actually needed. Thus, security breaches shall be prevented by only assigning these rights which are actually needed by a function. However, this is not always easy to fulfill since it is easier to give a function more rights than necessary if the system should work.

Thus, a simple test for the adequacy of test cases requires that each right assigned to a serverless function is at least used once. However, this requires a fine-granular hierarchy of potential access rights assignable to a function. Otherwise, if there is the possibility to grant a serverless function all rights being available, this single right can easily be covered by just calling the function once. Therefore, a hierarchy of assignable access rights has to be modelled, which is as coarse as needed but as fine-grained as possible.

This coverage criterion does not test functionality of a serverless application but the security. Missing access roles can be discovered while creating test cases for the fulfillment of the criterion, but the criterion indicates only if there are unused access rights.

In order to test the access right, the possibility to track the usage of the access rights is needed. This has either to be supported by the cloud platform provider or implemented manually. However, the latter one requires a deeper modification of the cloud platform and serverless application in order to observe the exact usage of resources.

3.4 Error handling

By moving the execution of the application to a cloud platform provider, the responsibility of handling errors moves to the platform provider, too. If an error occurs, the cloud platform provider usually tries to reexecute the function several times. However, the reexecution of a serverless function can be problematic if this function is not idempotent.

Not only errors caused by the logic of the function but also errors thrown by the cloud platform provider can occur. Two specific kinds of errors for a serverless function are the shortage of a resource (e.g., memory) used by the function and exceeding the time limit assigned to the serverless function. Since it cannot be guaranteed that these errors never occur for a function, it makes sense to test its behavior.

Therefore, we suggest the following coverage criteria:

– *All-restored* requires that each function fails at least once and is successfully reexecuted. Thus, the idempotency is tested.
– *All-failed* requires that every function fails at least once in each reexecution. Thus, the graceful degradation is tested.
– *All-failed-and-restored* requires both all-restored and all-failed to be fulfilled.

Only the serverless functions of the application need to be known for this criteria. However, the criteria can be extended if serverless functions are used in different contexts. By testing different contexts, a model displaying the relation or the workflow of the resources can support the coverage.

4 Future work

Based on our initial model and prototype tool, the model and its tool support will be extended to integrate and evaluate the different criteria discussed here in order to generate suitable sets of test cases for applications consisting of serverless functions. Additionally, the usage of mutation tests shall be investigated and applied for serverless applications.

5 Conclusion

This paper showed how the adequacy of integration test cases for serverless application can be tested. Classical approaches like control flow and data flow can be applied but also criteria focusing on certain characteristics of a serverless application by revealing certain classes of errors.

By applying the approaches suggested in this paper, test cases for serverless applications can be developed being less redundant and more focused on relevant error-prone characteristics. Thus, serverless applications with a better quality can be created more efficiently.

References

1. AWS Lambda. accessed March 11, 2019. URL: `https://aws.amazon.com/lambda/`.
2. Google Cloud Functions. accessed March 11, 2019. URL: `https://cloud.google.com/functions/`.
3. IBM Cloud Functions. accessed March 11, 2019. URL: `https://www.ibm.com/cloud/functions/`.
4. Azure Functions. accessed March 11, 2019. URL: `https://azure.microsoft.com/en-us/services/functions/`.
5. OpenFaaS. accessed March 11, 2019. URL: `https://github.com/openfaas`.

6. OpenLambda. accessed March 11, 2019. URL: `https://github.com/open-lambda`.
7. Ioana Baldini, Perry Cheng, Stephen J. Fink, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Philippe Suter, and Olivier Tardieu. The serverless trilemma: function composition for serverless computing. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software - Onward! 2017*. ACM Press, 2017.
8. Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing*, pages 1–20. Springer Singapore, 2017.
9. Gojko Adzic and Robert Chatley. Serverless computing: economic and architectural impact. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*. ACM Press, 2017.
10. Garrett McGrath and Paul R. Brenner. Serverless computing: Design, implementation, and performance. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2017.
11. Wei-Tsung Lin, Chandra Krintz, and Rich Wolski. Tracing function dependencies across clouds. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018.
12. Kerry Shih-Ping Chang and Stephen J. Fink. Visualizing serverless cloud application logs for program understanding. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2017.
13. Shang-Pin Ma, Chen-Yuan Fan, Yen Chuang, Wen-Tin Lee, Shin-Jie Lee, and Nien-Lin Hsueh. Using service dependency graph to analyze and test microservices. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, 2018.
14. Pamela Bhattacharya, Marios Iliofotou, Iulian Neamtiu, and Michalis Faloutsos. Graph-based analysis and prediction for software evolution. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012.
15. Magnus Madsen, Frank Tip, and Ondřej Lhoták. Static analysis of event-driven node.js JavaScript applications. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA 2015*. ACM Press, 2015.
16. M.E. Delamaro, J.C. Maidonado, and A.P. Mathur. Interface mutation: an approach for integration testing. *IEEE Transactions on Software Engineering*, 27(3):228–247, 2001.
17. Stefan Winzinger and Guido Wirtz. Model-based analysis of serverless applications. In *Proceedings of the 11th International Workshop on Modelling in Software Engineering - MiSE '19*, 2019.
18. Stefan Winzinger. Towards coverage criteria for serverless applications. In *11th Central European Workshop on Services and their Composition (ZEUS)*, 2019.
19. J. C. Huang. An approach to program testing. *ACM Computing Surveys*, 7(3):113–128, 1975.
20. U. Linnenkugel and M. Mullerburg. Test data selection criteria for (software) integration testing. In *Systems Integration '90. Proceedings of the First International Conference on Systems Integration*. IEEE Comput. Soc. Press, 1990.
21. Johannes Manner, Martin Endreß, Tobias Heckel, and Guido Wirtz. Cold start influencing factors in function as a service. In *Fourth International Workshop on Serverless Computing (WoSC4)*, Zurich, Switzerland, 2018.

22. Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Menezes Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. Cloud programming simplified: A berkeley view on serverless computing. Technical Report UCB/EECS-2019-3, EECS Department, University of California, Berkeley, 2019.

ProxiTour: A Smart Platform for Personalized Touring[⋆]

Aris Chronarakis[1], Stelios Gkouskos[2], Konstantinos Kalampokis[2], Gerasimos Papaioannou[2], Xenia Agalliadou[2], Ilias Chaldeakis[2] and Kostas Magoutis[1]

[1] Department of Computer Science and Engineering
University of Ioannina, Ioannina 45110, Greece
{achronarakis,magoutis}@cse.uoi.gr
[2] Terracom Informatics Ltd
79 Ethnikis Antistasis, Katsikas, Ioannina 45221, Greece
{sgous,kokalabo,ger.papaioannou,xagalliadou,ihaldeakis}@terracom.gr

Abstract. We describe a smart IoT-driven platform for personalized tours in indoor and outdoor spaces of cultural, touristic, or environmental interest. The platform is unique in its use of scalable data processing systems to process real-time IoT events coming from a commercial IoT platform, aiming to enhance user experience by intelligently interacting in a context-sensitive manner, within a personalized tour. In this paper we provide a preliminary evaluation of the stream processing component of the ProxiTour platform.

Keywords: Internet of Things · Context-awareness · Stream processing.

1 Introduction

We describe the ProxiTour platform, whose aim is to combine novel Internet-of-Things (IoT) and scalable data processing technologies to improve end-user experience when touring spaces of cultural, touristic and environmental interest. ProxiTour enhances user experience by taking into account the visitor's location in real time (a location based information platform) and enabling the concept of a personalized storyboard, while also taking into account personal user data such as age, interests, and dynamic behavior within the site. ProxiTour leverages historical information from past visits to detect patterns and deviations from usual behavior, and provide timely response to changes in user location.

The IoT devices being used in ProxiTour are Bluetooth beacons, wireless transducers detected by the Bluetooth stack of most mobile devices such as phones and tablets. These transmitters are placed in various places of interest (e.g., museums). Visitors that stay within the range of specific beacons for a certain period of time, receive information (visual and acoustic) on their mobile

[⋆] Supported by the Greek Research Technology Development and Innovation Action "RESEARCH - CREATE - INNOVATE", Operational Programme on Competitiveness, Entrepreneurship and Innovation (ΕΠΑνΕΚ) 2014-2020, Grant T1ΕΔΚ-04819 device related to the standing position and nearby exhibits. In outdoor spaces, ProxiTour can leverage location-tracking through GPS on mobile phones. The visitor's context, as well as that of the entire visitor population (historical information, as well as recent activity) is used to enrich the provided information through suggestions based on the visitor's profile, real-time position, and dynamic interaction with digital content provided by a mobile application.

The paper proceeds as follows. In Section 2 we discuss related work. In Section 3 we describe the architecture and implementation of ProxiTour, and in Section 4 a preliminary evaluation and current deployments of key ProxiTour components. Finally, in Section 5 we conclude and discuss future work.

2 Related work

Raw IoT sensor data become useful when we develop ways to collect, model, reason, and distribute its context information. Context-aware computing [1] refers to software, middleware, or service with the ability to gather and analyze information from its environment, and to provide this information to another service or to end users. Context information can be accessed according to the publish/subscribe communication pattern [2], where one or more users (subcribers) register their interest for specific information or condition (e.g. the value of an entity attribute exceeds a threshold), creating a group of interest. When such a condition is satisfied, information is published to the group.

Recent research has looked into leveraging IoT data and context-aware management technologies to build mobile-based solutions for customizing visits [3] or for analyzing collected data to learn how visitors choose to enter and spend time in the different rooms of a curated exhibition [4]. Previous works have focused on IoT-enabled mobile applications and on collecting and analyzing IoT data in an offline fashion. The ProxiTour platform differentiates from such works in that collected data can be analyzed in an online fashion as well, using stream-processing technologies, providing up-to-date statistics and near-real-time activity reports. Earlier work also proposed the use of stream-processing technologies for analyzing visitor positions tracked through cameras [5]. The present work extends prior work by leveraging Bluetooth beacons and by describing streaming analytics for determining, in near-real-time, the engagement of visitors with digital content served by a mobile application in a context-sensitive manner.

A popular context-aware computing platform is FIWARE [6], a EU joint venture with technology and telecommunications companies, aiming to improve know-how on Future Internet [7] applications and services. The FIWARE platform supports a range of Generic Enablers (GEs) [8], open-source general purpose software components that can be used as services through a user-friendly API. In FIWARE, the representation of information is based on the entity/attribute model. Each entity has a set of attributes, and a feature can have a set of metadata. The Orion Context Broker (CB) [9, 10] is a pub/sub style Context Broker available as a generic enabler within the FIWARE platform. Given the heterogeneity of sensor and other IoT device technologies, a bridging layer is needed between IoT devices and NGSI Context Brokers such as Orion. Such a functionality is typically implemented by IoT agents offering lightweight communication protocols, such as Ultralight 2.0 [11]. The transport protocols supported by IoT Agent UL2.0 between IoT devices and IoT Agent are HTTP and MQTT. Given that higher-level middleware such as stream- and data-processing platforms ingest data though systems such as Apache Kafka [12], a bridging layer between MQTT and Kafka is also needed for interoperability.

Processing high-volume streaming data is an important driver for creating business value in popular social networking and Internet services in general and has recently resulted into robust scalable systems (such as LinkedIn Samza [13], Twitter Heron [14], and Google Millwheel [15]) used in production. Computing click-through rates, evolving trends in social activity, and important operational metrics can increase service responsiveness, user engagement, and rapid response to operational issues. Stream processing systems have demonstrated high scalability and availability and the ability to rapidly adapt to higher load due to the need to re-process past data in addition to live traffic. In this paper we exhibit the use of a scalable stream processing engine in detecting events of interest within a personalized tour setting. Previous work in benchmarking stream-processing systems includes [16–18]. An evaluation of the fault-tolerance characteristics of the stream-processing platform used in ProxiTour was recently carried out [19].

3 Architecture and implementation

The ProxiTour architecture is modular and consists of a set of five subsystems whose structure and inter-communication (information flow) are depicted in Figure 1. The subsystems are the IoT platform, Content Management System (CMS), Mobile Application, Stream Processing Engine (SPE), and Storage and Big Data Analytics. Due to space constraints, in this paper we focus specifically on the IoT platform and the Stream Processing Engine (SPE).

Functionality of the ProxiTour system is driven by the continuous determination of the location of each visitor in space. In interior spaces where mobile devices cannot be detected by GPS, beacons are used to determine the approximate location of users (Figure 2). Beacons are small-sized devices and signal transmitters detectable by portable electronic devices (smartphones, tablets) within range (typically up to 70m). Beacons use the Bluetooth low energy (BLE) protocol to transport small amounts of data at low power consumption, maximizing their life (battery life of up to 2-3 years). The signal strength and the transmit interval between signals are adjustable to determine the desired coverage in each case (e.g., 1-2m radius). Entry of a visitor with a mobile device (smartphone, tablet) within the range of the beacon and for a certain period of time (adjustable parameter) triggers the transmission of information corresponding to that location (exhibit content) from the Content Management System (M2) to the visitor application (M3). This happens as follows: Detection of a beacon, in conjunction with its signal strength, by a mobile device gives an indication of the distance of the user (smartphone) from the beacon (*proximity* beacons). For

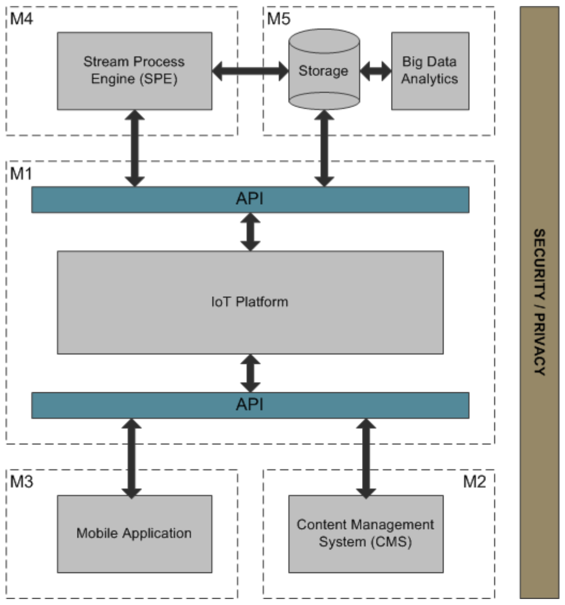

Fig. 1. ProxiTour Architecture

greater accuracy there is also the possibility of triangulation of signals from more beacons (*location* beacons). In order to properly cover spaces with beacons, an autopsy is typically carried out to decide the exact number of beacons required.

Information in the ProxiTour architecture flows as follows: The IoT platform (M1) periodically receives the approximate location (with an accuracy of about 1-1.5m) of a visitor inside the space (internal or external) from his mobile device (M3). Given up-to-date knowledge of *users (visitors)*, their *location* in the exhibit space, and *exhibits* that they currently digitally interact with, the IoT platform decides to serve appropriate exhibit content from the content management system (CMS) (M2) to users' mobile applications (M1). Simple behavior (such as what content to serve the first time a user is near an exhibit, when to refresh content, or how to respond to user interaction, such as clicking on specific fields) is specified within a simple rule base, part of M1. Additionally, the IoT platform forwards all information (user, location, interaction with exhibit(s)) to the M4 and M5 subsystems to feed into online and offline analytics and to obtain personalized information (e.g. exhibits that other visitors with similar interests spent significant time with), leading to personalized proposals for the visitor from the CMS (M2) (thematic routes, interest-based exhibits, etc.) to be displayed in the mobile application (M3). In Section 3.1 we describe streaming analytics determining which users are more engaged with the digital content, as well as popular exhibits in the recent past (e.g., last few minutes or hours).

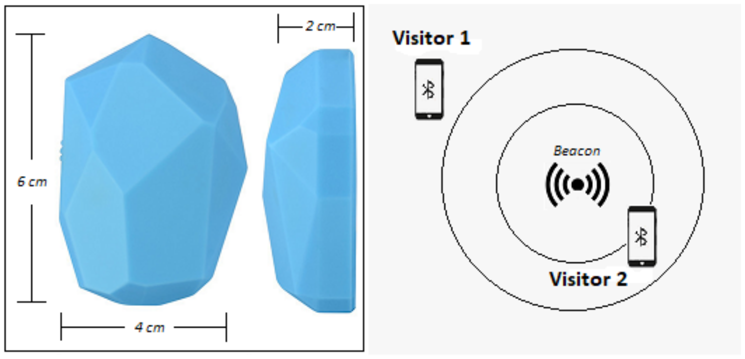

Fig. 2. Beacons

The IoT platform used in ProxiTour is Zastel[3], developed by Terracom Informatics Ltd to support the development of several commercial products, such as QR-Patrol[4], which is among the top real-time guard-tour security monitoring platforms worldwide with customers in 77 countries, and funded by the Horizon 2020 framework (SME Instrument Phase 2); and Spotypal[5], an anti-loss Bluetooth tracker and personal SOS system. With special emphasis placed on the interoperability features of Zastel with third platforms and systems, the past two years Zastel became compliant with two large European IoT platforms (symbIoTe and BIG IoT) and tested in pilot tests in providing a complete security solution for Universities and in monitoring traffic conditions in the city of Turin.

The basic functions of the IoT platform in ProxiTour are:

– Managing location services both internally (beacons) and externally (GPS)
– Dynamic data sharing and decision-making based on rules that have been set by site (e.g., museum) administrators and users
– Managing indoor beacons (battery levels, health, functionality, etc.) and administrative users
– Connecting with third-party systems and applications through an API

The IoT platform is managed by an administrator and a number of regular users. The administrator distributes user roles and sets simple rules governing system behavior (e.g. what content and when to serve when a user is first within (or re-enters) the range of an exhibit). Users receive online and offline analytics on visitor behavior, gaining a global view of how visitors interact with exhibits through the mobile application and digital content. Platform users are also able to manage IoT devices and to perform maintenance actions (such as tend to faulty beacons) when necessary.

The IoT platform (M1) runs on nodejs using RabbitMQ for interoperability with the rest of the system and MySQL as a backend database, as well as for the

[3] www.zastel.com
[4] www.qrpatrol.com
[5] www.spotypal.com

CMS (M2). ProxiTour solicits the explicit consent of users to collect data that they produce, respecting the privacy of such data when collected and stored (for example, through anonymization and/or encryption).

ProxiTour components M4 and M5 are implemented on top of the Apache Samza [13] platform. Samza is a *lambda-less* architecture, able to perform the processing of both real-time as well as batch data using the same dataflow structure. ProxiTour also includes a bridging layer between the MQTT protocol (RabbitMQ broker) used by the IoT platform and the Kafka [12] ingest engine used by the Apache Samza platform. The bridging layer acts as a MQTT consumer and a Kafka producer, relaying information between the two brokers.

3.1 Applications

We describe representative applications using stream analytics to derive knowledge about the degree of user engagement with museum exhibits and the exhibits that engage users most. These are important analytics for museum professionals, typically hard to compute in near-real-time as they require cross-correlation across information drawn from all visitors within specific time periods.

We assume that user devices are reporting contextual information about visitors (**UserID**), the location their devices are in (**LocationID**, e.g., coordinates in a grid mapping the museum floor), and exhibits that users are interacting with in their mobile platform (e.g., clicking on digital material) (**ExhibitID**). These tuples may contain null entries for the ExhibitID field, if a user is not currently interacting with digital material in its device. The tuples may contain null in the location field, if the device cannot localize itself (i.e., if it is out of range of beacons or GPS). A timestamp (indicating the time the information was produced) is also implicitly added by the mobile device as part of the tuple.

Degree of user engagement: This application takes as input the global stream of updates from all mobile devices and performs the following transformations:

- Select only tuples that have all fields set (i.e., drop nulls) and where device location is near (in a geometric sense) to the exhibit the user interacts digitally with
- Group tuples by UserID over window of time T. When windows close, compute number of exhibits each user interacted with in that period
- Collect previous tuples into another window operator, ranking (sorting) users according to their degree of interaction with exhibits
- The periodic output of this window is the list of *top-k* users and number (fraction) of engaged users over the total number of users

Exhbits that engage users most: This application takes as input the global stream of updates from mobile devices and performs the following transformations:

- Select only tuples that have all fields set (i.e., drop nulls) and where device location is near (in a geometric sense) to the exhibit the user interacts digitally with
- Group tuples by ExhibitID over window of time T. When windows close, count number of users that digitally interacted with each exhibit
- Collect previous tuples into another window operator, ranking (sorting) exhibits according to the degree of interaction with them
- Periodic output is the *top-k* exhibits over the chosen time period

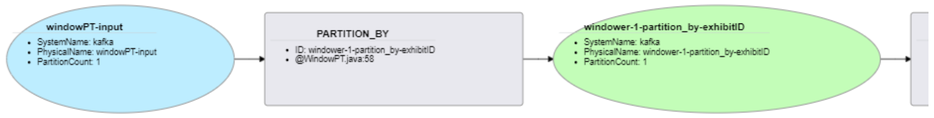
windowPT-input
SystemName: kafka
PhysicalName: windowPT-input
PartitionCount: 1
PARTITION_BY
ID: windower-1-partition_by-exhibitID
@WindowPT.java:58
windower-1-partition_by-exhibitID
SystemName: kafka
PhysicalName: windower-1-partition_by-exhibitID
PartitionCount: 1

Fig. 3. Find out exhibits that engage users most on their mobile app (left part)

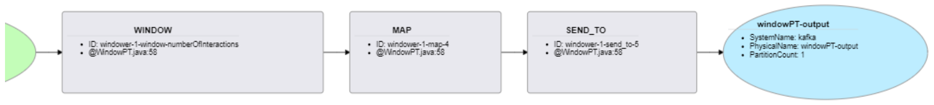

Fig. 4. Find out exhibits that engage users most on their mobile app (right part)

Implementation details Figures 3 (left part) and 4 (right part) depict the implementation of the second continuous (streaming) query described in Section 3.1 in Apache Samza. The input stream comprises events from all user mobile devices carrying information in the form **(UserID, LocationID, ExhibitID)**, represented as (key=user1, value={user1, location1, exhibit1}) in key-value form. The application partitions the input stream by the ExhibitID field (Figure 3) and writes it back to a Kafka topic with a single partition (PartitionCount: 1). An alternative implementation could increase the number of partitions to support a higher level of parallelism in the deployed application.

The partitioned information is read into a window operator grouping events by ExhibitID (Figure 4) counting interactions (event occurences) coming from all users with each specific exhibit within 5 sec time periods. Each time a window closes, the output tuple emitted has the form (exhibitID=exhbit1, interactions=6), etc. Sorting these outputs on the number of interactions in a subsequent operator yields the top-k currently most popular museum exhibits.

4 Evaluation

The ProxiTour platform is currently in an advanced development stage. The current evaluation focuses on individual components of the architecture using synthetic workloads. In Section 4.1 we provide information on a representative deployment of the IoT platform in a large factory and on the bridging layer between the IoT and data-processing subsystems. In Section 4.2 we evaluate the performance of the stream-processing module (M4) on synthetic sensor data modeled out of the information we anticipate to have as inputs in a museum pilot site.

4.1 IoT Platform

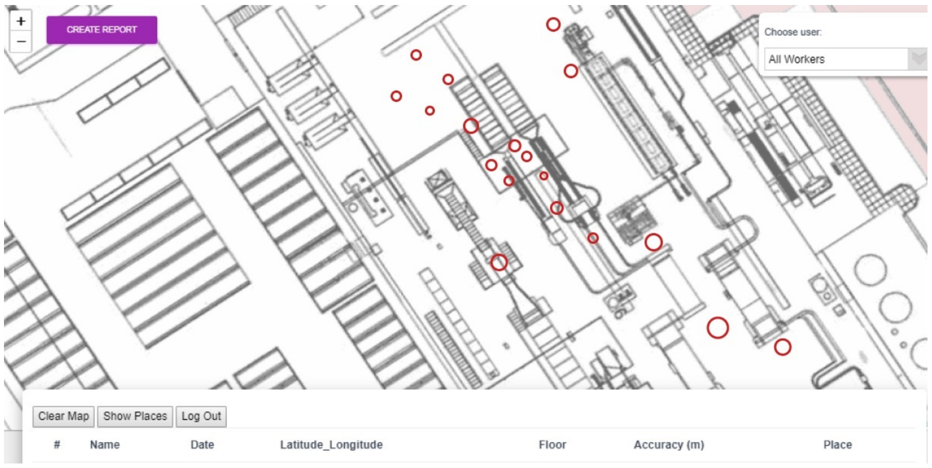

Fig. 5. Live map from indoor location platform (large factory pilot)

In the past few months the ProxiTour core IoT platform was trialed in a factory of more than 70 employees. With the use of a wearable BLE tag (transmitting every 100ms) and a dense matrix of 20 BLE gateways (transmitting every 2 sec) it was made possible to monitor the location and movements of workers with a precision of about 1-1.5m (Figure 5), which is satisfactory when projected to the ProxiTour targeted site of a medium to large-size museum. Since the installation of the IoT platform, more than 1GB of raw data are being collected on a daily basis. Their real-time processing offers upper management with an accurate view of operations as well as historical data. Zastel's storage structure supports different use-cases and has been proven to be a reliable and high performance platform under real-world conditions.

Our evaluation of the bridging layer between the MQTT protocol (RabbitMQ broker) used by the IoT platform and the Kafka ingest engine used by the SPE and big-data processing platform, indicates that we can comfortably transfer about 6,000 messages/sec between the two middleware layers over standard server/network infrastructure, considered sufficient for current ProxiTour requirements; further scalability is possible by setting up multiple broker paths between the two middleware layers.

4.2 Performance of the stream-processing system

In this evaluation we provide early results from the operation of the SPE module (M4), which is based on Apache Samza, under a representative load. In our evaluation we use the streaming analytics application of Figures 3 and 4 (Section 3.1) gauging the popularity of exhibits. Our experiments evaluate throughput in terms of number of input tuples processed for the application. Our experimental testbed consists of a cluster of servers each equipped with a Intel® Xeon Bronze® 3106 8-core CPU clocked at 1.70GHz, 16GB DDR4 2666MHz DIMMs, and a 256GB Intel SSD, running Ubuntu Linux 16.04.6 LTS. The nodes are interconnected through a 10Gb/s Dell N4032 switch. The software versions used are Samza version 1.1.0 (latest stable version at the time of this work), Kafka version 0.10.1.1, Zookeeper 3.4.3, and Yarn version 2.6.1.

The streaming application uses the Samza high level API. In these experiments, each input stream maps to a single partition of a Kafka topic. We use a single Samza instance within a container deployed in a single-node Yarn implementation. The benchmark setup is deployed on two servers. One server (node01) hosts the Kafka broker and Kafka producers feeding the input topic with tuples, and the ZooKeeper service used by Kafka. The second server (node02) hosts the single-node Samza container. We measure CPU usage and throughput on both machines. The workload generator consists of 3 concurrent tasks producing a 3 million (M) tuples each, for a total of 9M tuples. The tuple keys are drawn uniformly at random from a set of 10 distinct keys. Since the continuous query writes an intermediate topic (9M tuples), Samza reports to read a total of 18M tuples from Kafka.

Each topic partition has an offset which indicates the tuple from input topics that should be consumed next. In case of many partitions of a topic, each partition has a separate offset. We use periodic readings of this offset to compute our throughput metric, measuring how many tuples are consumed at specific intervals by the stream processor. Tables 1 and 2 summarize our results over two runs. A general observation is that workloads are more resource-demanding at the workload generator (spending 50% of the CPU on average) while Samza easily achieves tuple processing rates of about 30K/sec.

	node01 (workload generator)	node02 (Samza stream processor)
CPU busy (%)	50	15

Table 1. Average CPU spent under the popular-exhibits application

	Messages actually processed (tuples/sec)
Experiment 1 (avg)	29,364
Experiment 2 (avg)	29,460

Table 2. Throughput under the popular-exhibits application

5 Conclusions and future work

In this paper we introduced the architecture and selected key components of ProxiTour, a smart IoT-driven platform for personalized tours in indoor and outdoor spaces of cultural, touristic, or environmental interest. We provided a high level view and information from a representative deployment of the ProxiTour IoT platform in an industrial environment. We described a use-case based on a continuous query evaluating the digital interaction of visitors with exhibits. Our preliminary evaluation of the stream-processing engine shows that it can easily support sufficiently high event rates. These first results indicate that the ProxiTour architecture has the potential to collect and analyze a wealth of information about visitors and their digital interaction with exhibits in an online, near-real-time fashion. Our future work will focus in further integrating the components of the ProxiTour platform, in extending our range of analytics, and in enhancing user experience through personalized recommendations taking into account a timely global view of visitor profiles, personal interests, and up-to-date statistics of visitor interaction with exhibits.

6 Acknowledgements

The work reported in this paper was performed in the context of the project "ProxiTour: A Platform for Personalized Touring". The authors thankfully acknowledge funding by the Greek Research Technology Development and Innovation (RTDI) Action "RESEARCH - CREATE - INNOVATE", Operational Programme on Competitiveness, Entrepreneurship and Innovation (ΕΠΑνΕΚ) 2014-2020; Grant ID: Τ1ΕΔΚ-04819.

References

1. C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys and Tutorials*, 16(1):414–454, 2014.
2. P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.
3. I. Ayala, M. Amor, M. Pinto, L. Fuentes, and N. Gámez. iMuseumA: an agent-based context-aware intelligent museum system. *Sensors (Basel, Switzerland)*, 14(11):21213–21246, 2014.
4. R. Pierdicca, M. Marques-Pita, M. Paolanti, and E. S Malinverni. IoT and Engagement in the Ubiquitous Museum. *Sensors (Basel, Switzerland)*, 19(6):1387, 2019.

5. D. Stamatakis, D. Grammenos, and K. Magoutis. Real-Time Analysis of Localization Data Streams for Ambient Intelligence Environments. In *Proc. of 2nd Int. Conf. on Ambient Intelligence (AmI'11)*, pages 92–97, Amsterdam, The Netherlands, 2011.
6. FIWARE Platform. https://www.fiware.org.
7. Future Internet. https://www.ict-fire.eu/.
8. FIWARE Generic Enablers Catalogue. https://catalogue.fiware.org/enablers.
9. Orion Context Broker. https://github.com/telefonicaid/fiware-orion.
10. Orion Context Broker Documentation. https://fiware-orion.readthedocs.io.
11. IoT Agent UL2.0. https://github.com/telefonicaid/iotagent-ul.
12. J. Kreps, N. Narkhede, and J. Rao. Kafka: A distributed messaging system for log processing. In *Proc. NetDB*, page 1–7, Athens, Greece, June 12, 2011.
13. S. Noghabi, K. Paramasivam, Y. Pan, N. Ramesh, J. Bringhurst, I. Gupta, and R. H. Campbell. Samza: Stateful Scalable Stream Processing at LinkedIn. In *Proceedings of the VLDB Endowment, Vol. 10, No. 12*, 2017.
14. S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. Patel, K. Ramasamy, and S. Tanej. Twitter Heron: Stream processing at scale. In *In Proceedings of SIGMOD*, 2015.
15. T. Akidau, A. Balikov, K. Bekiroglu, S. Chernyak, J. Haberman, et al. Millwheel: fault-tolerant stream processing at internet scale. In *Proc. VLDB, pages 1033–1044*, 2013.
16. J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, and V. Markl. Benchmarking distributed stream processing engines. *CoRR*, abs/1802.08496, 2018.
17. S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. Peng, and P. Poulosky. Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming. In *Proc. of IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW 2016)*, pages 1789–1792, Los Alamitos, CA, USA, may 2016. IEEE Computer Society.
18. Z. Zhuang, T. Feng, Y. Pan, H. Ramachandra, and B. Sridharan. Effective multi-stream joining in apache samza framework. In Calton Pu, Geoffrey C. Fox, and Ernesto Damiani, editors, *BigData Congress*, pages 267–274. IEEE Computer Society, 2016.
19. A. Chronarakis, A. Papaioannou, and K. Magoutis. On the impact of log compaction on incrementally checkpointing stateful stream-processing operators. In *Proc. of 1st Workshop on Distributed and Reliable Storage Systems (DRSS'19), held in conjunction with SRDS'19*, Lyon, France, Oct. 1, 2019.

A Survey on Cloud Migration Strategies for High Performance Computing

Stefan Kehrer and Wolfgang Blochinger

Parallel and Distributed Computing Group, Reutlingen University, Alteburgstr. 150, 72762 Reutlingen, Germany
`[firstname.lastname]@reutlingen-university.de`

Abstract. The cloud evolved into an attractive execution environment for parallel applications from the High Performance Computing (HPC) domain. Existing research recognized that parallel applications require architectural refactoring to benefit from cloud-specific properties (most importantly elasticity). However, architectural refactoring comes with many challenges and cannot be applied to all applications due to fundamental performance issues. Thus, during the last years, different cloud migration strategies have been considered for different classes of parallel applications. In this paper, we provide a survey on HPC cloud migration research. We investigate on the approaches applied and the parallel applications considered. Based on our findings, we identify and describe three cloud migration strategies.

Keywords: Cloud Computing · High Performance Computing · Parallel Application · Cloud Migration · Cloud-aware Refactoring ·

1 Introduction

Traditionally, many parallel applications have been designed and developed for HPC clusters. However, more recently, the cloud evolved into an attractive execution environment for HPC workloads [26, 38]. Former research on this topic mainly investigates how to make cloud environments HPC-aware [25]. In particular, resource pooling and virtualization leading to heterogeneous processing speeds as well as low network throughput and high network latency have been addressed [8, 41]. During the last years great progress has been made with respect to HPC-aware cloud environments. As of today, many cloud providers, including Amazon Web Services (AWS)[1] and Microsoft Azure[2], offer cloud environments optimized for HPC workloads [44, 1].

On the other hand, there is a growing interest to make parallel applications cloud-aware [12, 13, 34, 30]. However, this requires architectural refactoring, which comes with many challenges and cannot be applied to all applications due to performance issues [19, 5]. As a result, different cloud migration strategies have been applied to different parallel applications.

[1] https://aws.amazon.com.
[2] https://azure.microsoft.com.

As more and more research considers the migration of parallel applications to the cloud, we argue that a survey on HPC cloud migration research is required to understand current research issues. In this paper, we investigate on HPC cloud migration and describe three cloud migration strategies identified in existing research. The remainder of this paper is structured as follows. In Section 2, we describe two different types of cloud environments that can be employed to operate parallel applications. In Section 3, we present the research method and search strategy underlying our survey. Based on existing research, we describe the HPC cloud migration strategies identified in Section 4. Moreover, we discuss the key findings of our survey. In Section 5, we conclude our work and describe future research opportunities.

2 Cloud Environments for HPC

Two different types of cloud environments can be employed to operate parallel applications: Standard and HPC-aware cloud environments. Standard cloud environments often suffer from CPU time sharing leading to heterogeneous processing speeds as well as low network throughput and high network latency, which are well-known effects of resource pooling and virtualization [8, 41]. On the other hand, HPC-aware cloud environments limit these negative side-effects by means of the following concepts:

– CPU affinity: HPC-aware cloud environments ensure CPU affinity both at the application and at the hypervisor level. As a result, vCPUs are mapped to physical CPU cores leading to improved cache locality and higher cache hit rates [13]. This concept is also referred to as CPU pinning.
– HPC-aware virtual machine (VM) placement policies: Standard cloud schedulers do not ensure co-location of VMs. Existing work has shown that HPC-aware VM placement effectively resolves this issue and leads to significant performance gains [14].
– Guaranteed network performance: HPC-aware cloud environments are based on novel concepts such as single root input/output virtualization (SR-IOV) to ensure guaranteed network bandwidth and latency with constant quality of service while supporting resource pooling and network virtualization [25].
– Disabled VM migration: Typically, VM migration is disabled to avoid environmental overhead.
– Disabled memory overcommitment: Memory overcommitment leads to preemption and paging and is typically disabled [7].
– Lightweight virtualization: Container-based virtualization (OS-level virtualization) ensures lower overheads compared to hypervisor-based virtualization [43].

3 Research Method and Search Strategy

Our survey is based on a literature review to identify existing research on HPC cloud migration. We followed the steps of a literature review process described in [3]. For the search process, we employed the ACM Digital Library[3], IEEE Xplore[4], and Google Scholar[5]. The following search query was used for the search: *(”cloud” OR ”elastic” OR ”elasticity”) AND (”migrat*” OR ”transform” OR ”convert” OR ”refactor*” OR ”adapt*” OR ”modification”) AND (”hpc” OR ”parallel application”).*

We selected relevant articles based on a manual selection process: Only peer-reviewed articles (published in English) have been included. Moreover, only articles that discuss in detail how to migrate parallel applications to the cloud were selected. Additional literature has been identified (1) by reviewing the references of selected articles and (2) by analyzing the citations of these articles [40].

4 Cloud Migration Strategies for HPC

In this section, we review existing work on HPC cloud migration. This section is structured according to the three cloud migration strategies that we have identified: (1) *Copy & Paste*, (2) *Cloud-aware Refactoring*, and (3) *Cloud-aware Refactoring & Elasticity Control*. Table 1 summarizes our classification of existing work. For each cloud migration strategy, we describe the key findings in detail.

4.1 Copy & Paste

This migration strategy proposes the migration of existing parallel applications without modifications. Both standard and HPC-aware cloud environments have been evaluated by following this migration strategy. For instance, the authors of [6] evaluate an existing application based on the Message Passing Interface (MPI) [10] deployed to a standard cloud environment. Moreover, serial versions of the NAS Parallel Benchmarks (NPB) [2] have been employed to assess the computational performance of different instance types. The results obtained show that the lower performance measured is mainly related to the high network latencies and low network bandwidths in standard cloud environments. The authors of [33] also evaluate existing MPI-based applications from the NPB in standard cloud environments. On the other hand, the authors of [23] investigate MPI-based applications in HPC-aware cloud environments and measure low variance in network bandwidth. Moreover, the raw computation performance has been shown to be comparable to HPC clusters, even if virtualization overhead exists. The authors of [42] consider HPC-aware cloud environments more cost-effective compared to traditional HPC clusters if a cluster does not achieve high utilization. The authors of [12] evaluate different parallel applications based on MPI and CHARM++ deployed to both standard and HPC-aware cloud environments. Whereas specifically tightly-coupled applications suffer from the low network bandwidth and high network latency in standard cloud environments,

[3] https://dl.acm.org.
[4] https://ieeexplore.ieee.org.
[5] https://scholar.google.com.

Table 1. This table shows our classification of existing work on HPC cloud migration.

Selected Article	**Cloud Environment**	**Elasticity Control**	**Cloud Migration Strategy**
Evangelinos et al. [6]	Standard	None	*Copy & Paste*
Roloff et al. [33]	Standard	None	*Copy & Paste*
Marathe et al. [23]	HPC-aware	None	*Copy & Paste*
Zhai et al. [42]	HPC-aware	None	*Copy & Paste*
Gupta et al. [12]	Standard / HPC-aware	None	*Copy & Paste*
Rajan et al. [28]	HPC-aware	None	*Copy & Paste*
Gupta et al. [15]	Standard	None	*Cloud-aware Refactoring*
Rajan et al. [30]	Standard	None	*Cloud-aware Refactoring*
Vu et al. [39]	Standard	None	*Cloud-aware Refactoring*
Kehrer et al. [20]	Standard	None	*Cloud-aware Refactoring*
Da Rosa Righi et al. [34]	Standard	Reactive	*Cloud-aware Refactoring & Elasticity Control*
Da Rosa Righi et al. [35]	Standard	Reactive	*Cloud-aware Refactoring & Elasticity Control*
Rodrigues et al. [32]	Standard	Hybrid	*Cloud-aware Refactoring & Elasticity Control*
Da Rosa Righi et al. [36]	Standard	Hybrid	*Cloud-aware Refactoring & Elasticity Control*
Raveendran et al. [31]	Standard	Reactive	*Cloud-aware Refactoring & Elasticity Control*
Rajan et al. [28]	Standard	Reactive	*Cloud-aware Refactoring & Elasticity Control*
Haussmann et al. [16]	Standard	Reactive	*Cloud-aware Refactoring & Elasticity Control*

HPC-aware cloud offerings have been shown to effectively overcome these issues. Thus, tightly-coupled applications benefit from on-demand access to compute resources and the freedom to select the number of processing units. Having the freedom to select the number of processing units per application run is new to HPC users as traditionally the number of processing units is limited by resource quotas or one tries to optimize the number of processing units to get a job scheduled faster (e.g., in HPC clusters with job schedulers).

Whereas most approaches require the manual selection of the number of processing units, recent work also shows how to automatically select the number of processing units in an HPC-aware cloud environment when the computational steps and communication patterns of the application can be captured in form of an *application model*. Based on the application model, one is able to calculate how the number of processing units effects execution time, speedup, efficiency, and monetary costs thus allowing versatile optimizations per application run. The authors of [28] specifically address applications based on the split-map-merge paradigm and consider the cost-time product as objective function to statically select the optimal number of processing units per application run. Therefore, the automated selection process considers (1) information on the input problem, (2) an application model built for split-map-merge applications, (3) a user-defined objective function (in this case the cost-time product as a function of the number of processing units), and (4) information on the execution environment (e.g., processing speed, network bandwidth) obtained by measuring sample workloads.

By following this approach, parallel applications benefit from on-demand access to compute resources and pay-per-use, which enables fine-grained cost control per application run. Because the number of processing units does not have to be adapted at runtime, an existing parallel application can be deployed to an HPC-aware cloud environment without modifications.

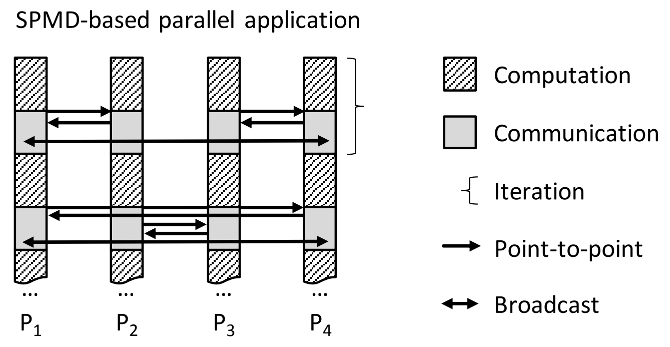

Fig. 1. Many parallel applications are developed based on the Single Program Multiple Data (SPMD) application model and rely on synchronous communication in globally defined communication phases.

Findings: In general, many existing parallel applications are implemented based on the Single Program Multiple Data (SPMD) application model [22, 24, 25] (especially supported by MPI) and rely on frequent synchronous communication. A prototypical SPMD-based MPI application is given in Fig. 1. In each iteration, every MPI process executes local computations. Thereafter, a pair of MPI processes exchanges application-specific data via point to point communication. After all process pairs finished their data transfers, updates required globally are sent to other processes by means of an MPI broadcast primitive (cf. Fig. 1). For tightly-coupled SPMD-based parallel applications, HPC-aware cloud offerings provide an execution environment that can be used analogously to an HPC cluster but allows individual configuration of compute resources (by means of virtualization techniques). In existing work, elasticity is not employed and thus the number of processing units has to be statically selected. However, by following the *Copy & Paste* migration strategy, parallel applications benefit from an on-demand provisioned execution environment that can be payed on a per-use basis and individual configuration of compute resources.

4.2 Cloud-aware Refactoring

This cloud migration strategy proposes architectural refactoring to make existing parallel applications cloud-aware [27, 19]. We discuss several examples for cloud-aware refactoring in the following.

The authors of [15] introduce a dynamic load balancing mechanism to address the challenges of tightly-coupled iterative MPI-based applications in standard cloud environments caused by virtualization and resource pooling. Therefore, a dynamic load balancer continuously monitors the load of each vCPU and reacts to a measured imbalance by adapting the task distribution to virtual machines. Task overdecomposition is used to enable dynamic load balancing, which effectively reduces idle time.

The authors of [30] employ the Work Queue framework [4] to develop parallel applications for standard cloud environments. The Work Queue framework is designed for scientific ensemble applications and provides a master/worker architecture with an elastic pool of workers. The application employed in the presented case study is designed for replica exchange molecular dynamics (REMD) and can be considered as iterative-parallel. Similar applications are discussed by the authors in [29].

The authors of [39] employ standard cloud environments to operate irregular task-parallel applications and present a work stealing algorithm that selects victims (other processing units) based on the measured network link latency. Processing units with a lower latency are preferred for stealing operations. The presented algorithm is self-adaptive and has been shown to outperform other load balancing mechanisms that do not consider network link latency.

The authors of [20] specifically address applications with dynamic task parallelism and discuss how these applications can benefit from elastic scaling. They argue that, in cloud environments, parallel applications have to dynamically adapt the degree of logical parallelism based on a dynamically changing physical parallelism, given by the number of processing units (e.g., number of vCPUs, VMs). Based on their findings, the authors describe the design and implementation of a cloud-aware runtime system for elastic task-parallel processing in the cloud. The presented runtime system transparently controls the parallelism of an application to ensure elastic scaling. Therefore, developers mark parallelism in the program and the runtime system automatically adapts the logical parallelism by generating tasks whenever required. The runtime system exploits available processing units with maximum efficiency by mapping the logical parallelism (tasks) to the physical parallelism (processing units). An application based on this cloud-aware runtime system is elastically scalable because newly provisioned processing units (VMs) automatically receive tasks by means of load balancing and a task migration mechanism releases processing units that have been selected for decommissioning. To decouple task generation and task processing, the runtime system is based on the distributed task pool execution model and solves parallel coordination problems based on Apache ZooKeeper[6]. The authors state that the runtime system is not limited to any specific cloud management approach or tooling: Cloud management may comprise any kind of external decision making logic that finally adapts the number of processing units (i.e., the physical parallelism).

Findings: Cloud-aware refactoring has been specifically employed in the context of standard cloud environments. Typically, heterogeneous processing speeds and varying network latencies negatively effect parallel applications that employ synchronous communication and / or barrier synchronization. Cloud-aware refactoring can be employed to make these applications less affected by the characteristics of standard cloud environments. By following this migration strategy, one is able to exploit (low cost) standard cloud resources while still maximizing parallel performance. However, it has also been recognized that such an approach cannot be applied to all applications. Specifically, in the context of parallel applications with frequent communication and synchronization, architectural refactoring leads to fundamental performance issues and thus cannot be applied.

4.3 Cloud-aware Refactoring & Elasticity Control

This cloud migration strategy proposes the use of elasticity to process HPC workloads in the cloud. Therefore, architectural refactoring of parallel applications is fundamentally required to deal with a varying number of processing units. In the following, we discuss several examples for this migration strategy and describe the elasticity control mechanisms considered.

The authors of [34] describe a reactive elasticity control mechanism for iterative-parallel applications. The presented concept called AutoElastic supports the automated transformation (source-to-source translation) of existing MPMD[7]-based

[6] https://zookeeper.apache.org.
[7] Multiple Program Multiple Data.

MPI-2 applications with a master/worker architecture into elastic parallel applications. MPI-2 features dynamic process management and thus supports a varying number of MPI processes [11]. The presented concepts are evaluated by using a numerical integration application which simulates different dynamic workload patterns (e.g., ascending, descending, and wave workload). Similar concepts are discussed in [35].

The authors of [32] present a hybrid elasticity controller based on a technique called live thresholding, which has also been used in [36]. Live thresholding dynamically adapts the thresholds of a reactive elasticity controller, which is implemented as a closed feedback-loop architecture. Workload patterns are detected by comparing the last two average load values calculated based on monitored time series data and simple exponential smoothing. Similar concepts are discussed in [36]. Both approaches address iterative-parallel applications.

The authors of [31] propose a concept to transform MPI-based iterative-parallel applications into elastic applications. They describe how to adapt existing MPI-based applications to deal with a dynamically changing number of processing units. The presented approach basically terminates a running application and restarts the application with a different number of processing units. Termination can only be applied at certain points in the program, e.g., at the end of an iteration. The described elasticity controller is designed to optimize the desired execution time, which is estimated based on the number of iterations and the average iteration time. The underlying assumption is that the amount of work per iteration is constant. Scaling decisions are made by comparing the measured average iteration time with the required iteration time to complete within the user-defined execution time: If the average iteration time is below the required iteration time, processing units are added. Otherwise, processing units are removed.

The authors of [28] (we already discussed this work in Section 4.1) also describe a second approach to use their application model: Whenever the characteristics of the cloud environment (e.g., processing speed, network bandwidth) are expected to change at runtime, an elasticity controller monitors the environment and continuously evaluates the objective function based on monitoring data. When the optimal (with respect to the user-defined objective function) number of processing units changes, the elasticity controller dynamically adapts the resource configuration. A cloud-aware application architecture is required to support such adaptations at runtime.

The authors of [16] discuss elasticity-related opportunities and challenges for irregular task-parallel applications. Their computation and communication patterns are input-dependent, unstructured, and evolving during the computation and thus their runtime and scaling behavior cannot be determined upfront [9, 37]. As a result, the total number of essential basic computational steps per time unit is unknown a priori and cannot be predicted. These applications comprise an unpredictable workload pattern. The authors discuss the two conflicting objectives of fast processing and low monetary costs finally leading to a multi-objective optimization problem and Pareto optimal solutions, which prevents automated decision making with respect to the number of processing units. To deal with this problem, the authors employ the concept of opportunity costs to convert the underlying objective functions into a single aggregated objective function, thus allowing cost-based selection of the number of processing units. Because one cannot reason about the effects on execution time, speedup, efficiency, and monetary costs in absolute terms, the authors present a reactive elasticity controller for heuristic cost optimization: The cost function is approximated based on metrics monitored at runtime. Therefore, the elasticity controller continuously monitors the application and evaluates the defined objective function (minimize the monetary costs based on the presented cost model). The authors empirically evaluate their elasticity controller by comparing the results of their heuristic cost optimization approach with the minimum monetary costs (which can be obtained by measuring the scalability of the application with exemplary input problems).

Findings: This migration strategy proposes the use of elasticity to deal with dynamic and unpredictable workload patterns and / or environmental changes in standard cloud environments. Either proactive, reactive, or hybrid elasticity control mechanisms can be employed (depending on the characteristics of the application). Additionally, a cloud-aware application architecture is fundamentally required to ensure that a parallel application dynamically adapts to a changing number of processing units (selected by an elasticity controller). By following this migration strategy, parallel applications benefit from elasticity in form of more efficiently employed compute resources.

4.4 Summary and Discussion

Specifically for the *Copy & Paste* migration strategy, the ongoing evolution of HPC-aware cloud environments provides attractive benefits when compared to traditional HPC cluster environments. Existing work following this migration strategy does not make use of elasticity. Because mainly tightly-coupled data-parallel applications have been considered, this can be explained by considerable repartitioning efforts (when processing units are added or removed). With the technology available today, it is not possible for tightly-coupled SPMD-based applications to make use of elasticity to optimize costs and efficiency by adding or removing processing units during the computation due to the high overheads that would result from repartitioning.

The *Cloud-aware Refactoring* migration strategy proposes architectural refactoring of existing parallel applications. Architectural refactoring, in general and specifically in the context of parallel applications, is a comparatively new concept. The authors of [45] applied architectural refactoring to develop cloud-native applications. In [19], we present an approach to assess the cloud readiness of parallel applications that can be used to gain insights into the architectural changes required. In this context, we also recognized that many parallel applications provide several degrees of freedom with respect to their architecture.

Finally, we identified a third migration strategy: *Cloud-aware Refactoring & Elasticity Control*. Existing work following this migration strategy focuses on the benefits that can be obtained by means of elasticity. Specifically, in [28] and

[16], monetary costs and time are explicitly considered. The different approaches described basically result from the different characteristics of the applications addressed: Whereas the scaling behavior of applications based on the split-map-merge paradigm can be predicted based on an application model [28], the scaling behavior of irregular task-parallel applications is unknown upfront and unpredictable by nature, which requires reactive elasticity control [16]. On the other hand, both approaches convert the underlying multi-objective optimization problem into a single-objective optimization problem to enable automated decision making with respect to the optimal number of processing units. The authors of [28] use the cost-time product to create a single-objective optimization problem. The authors of [16] employ the concept of opportunity costs, which can be used to express time in terms of costs, thus enabling a purely cost-driven optimization.

5 Conclusion

With the aim of providing contributions to practitioners and researchers alike, we present a classification of HPC cloud migration research and describe the key findings. Most importantly, we recognized that elasticity, which is often considered to be the fundamental property of cloud environments, can only be beneficially employed under certain circumstances. More research is required to understand elasticity-related opportunities and challenges in the context of HPC. Whereas HPC users traditionally had no visibility of the monetary costs of compute resources in cluster environments, the pay-per-use property requires users to consider costs in cloud environments. Related work shows how to exploit on-demand compute resources and elasticity to control the monetary costs of parallel computations in the cloud. These approaches dynamically adapt the number of processing units under consideration of scare resources such as time and money. However, as of today, a clear and generally applicable understanding of elasticity in the context of HPC does not exist.

Our long-term goal is to understand how to design, develop, and manage cloud-aware parallel applications, i.e., applications that leverage cloud-specific properties such as on-demand access to compute resources, pay-per-use, and elasticity. Therefore, we follow a multi-faceted approach by investigating design-level, programming-level, and system-level aspects [19, 20] as well as delivery and deployment automation [18, 17, 21].

Acknowledgements. This research was partially funded by the Ministry of Science of Baden-Württemberg, Germany, for the Doctoral Program *Services Computing*.

References

1. Aljamal, R., El-Mousa, A., Jubair, F.: A comparative review of high-performance computing major cloud service providers. In: 2018 9th International Conference on Information and Communication Systems (ICICS). pp. 181–186 (April 2018)

2. Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Dagum, L., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Schreiber, R.S., Simon, H.D., Venkatakrishnan, V., Weeratunga, S.K.: The nas parallel benchmarks summary and preliminary results. In: Supercomputing '91:Proceedings of the 1991 ACM/IEEE Conference on Supercomputing. pp. 158–165 (Nov 1991)
3. Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M.: Lessons from applying the systematic literature review process within the software engineering domain. Journal of Systems and Software **80**(4), 571 – 583 (2007)
4. Bui, P., Rajan, D., Abdul-Wahid, B., Izaguirre, J., Thain, D.: Work queue+python: A framework for scalable scientific ensemble applications. In: Workshop on Python for High-Performance and Scientific Computing (2011)
5. Ekanayake, J., Fox, G.: High performance parallel computing with clouds and cloud technologies. In: Avresky, D.R., Diaz, M., Bode, A., Ciciani, B., Dekel, E. (eds.) Cloud Computing. pp. 20–38. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
6. Evangelinos, C., Hill, C.N.: Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazons ec2. In: In The 1st Workshop on Cloud Computing and its Applications (CCA (2008)
7. Ferreira, K.B., Bridges, P., Brightwell, R.: Characterizing application sensitivity to os interference using kernel-level noise injection. In: 2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–12 (Nov 2008)
8. Galante, G., Erpen De Bona, L.C., Mury, A.R., Schulze, B., da Rosa Righi, R.: An analysis of public clouds elasticity in the execution of scientific applications: a survey. Journal of Grid Computing **14**(2), 193–216 (Jun 2016)
9. Gautier, T., Roch, J.L., Villard, G.: Regular versus irregular problems and algorithms. In: Ferreira, A., Rolim, J. (eds.) Parallel Algorithms for Irregularly Structured Problems. pp. 1–25. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
10. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: portable parallel programming with the message-passing interface. MIT press, third edn. (2014)
11. Gropp, W., Thakur, R., Lusk, E.: Using MPI-2: Advanced features of the message passing interface. MIT press (1999)
12. Gupta, A., Faraboschi, P., Gioachin, F., Kale, L.V., Kaufmann, R., Lee, B., March, V., Milojicic, D., Suen, C.H.: Evaluating and improving the performance and scheduling of hpc applications in cloud. IEEE Transactions on Cloud Computing **4**(3), 307–321 (July 2016)
13. Gupta, A., Kale, L.V., Gioachin, F., March, V., Suen, C.H., Lee, B.S., Faraboschi, P., Kaufmann, R., Milojicic, D.: The who, what, why, and how of high performance computing in the cloud. In: IEEE 5th International Conference on Cloud Computing Technology and Science. vol. 1, pp. 306–314 (Dec 2013)
14. Gupta, A., Kal, L.V., Milojicic, D., Faraboschi, P., Balle, S.M.: Hpc-aware vm placement in infrastructure clouds. In: 2013 IEEE International Conference on Cloud Engineering (IC2E). pp. 11–20 (March 2013)
15. Gupta, A., Sarood, O., Kale, L.V., Milojicic, D.: Improving hpc application performance in cloud through dynamic load balancing. In: 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. pp. 402–409 (May 2013)
16. Haussmann, J., Blochinger, W., Kuechlin, W.: Cost-efficient parallel processing of irregularly structured problems in cloud computing environments. Cluster Computing (Dec 2018)

17. Kehrer, S., Blochinger, W.: Autogenic: Automated generation of self-configuring microservices. In: Proceedings of the 8th International Conference on Cloud Computing and Services Science. pp. 35–46. SciTePress (2018)
18. Kehrer, S., Blochinger, W.: Tosca-based container orchestration on mesos. Computer Science - Research and Development **33**(3), 305–316 (Aug 2018)
19. Kehrer, S., Blochinger, W.: Migrating parallel applications to the cloud: assessing cloud readiness based on parallel design decisions. SICS Software-Intensive Cyber-Physical Systems **34**(2), 73–84 (Jun 2019)
20. Kehrer, S., Blochinger, W.: Taskwork: A cloud-aware runtime system for elastic task-parallel hpc applications. In: Proceedings of the 9th International Conference on Cloud Computing and Services Science. pp. 198–209. SciTePress (2019)
21. Kehrer, S., Riebandt, F., Blochinger, W.: Container-based module isolation for cloud services. In: 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE). pp. 177–186 (2019)
22. Keutzer, K., Massingill, B.L., Mattson, T.G., Sanders, B.A.: A design pattern language for engineering (parallel) software: merging the plpp and opl projects. In: Proceedings of the 2010 Workshop on Parallel Programming Patterns. ACM (2010)
23. Marathe, A., Harris, R., Lowenthal, D.K., de Supinski, B.R., Rountree, B., Schulz, M., Yuan, X.: A comparative study of high-performance computing on the cloud. In: Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing. pp. 239–250. HPDC '13, ACM, New York, NY, USA (2013)
24. Massingill, B.L., Mattson, T.G., Sanders, B.A.: Reengineering for parallelism: an entry point into plpp for legacy applications. Concurrency and Computation: Practice and Experience **19**(4), 503–529 (2007)
25. Mauch, V., Kunze, M., Hillenbrand, M.: High performance cloud computing. Future Generation Computer Systems **29**(6), 1408 – 1416 (2013)
26. Netto, M.A.S., Calheiros, R.N., Rodrigues, E.R., Cunha, R.L.F., Buyya, R.: Hpc cloud for scientific and business applications: Taxonomy, vision, and research challenges. ACM Computing Surveys (CSUR) **51**(1), 8:1–8:29 (Jan 2018)
27. Parashar, M., AbdelBaky, M., Rodero, I., Devarakonda, A.: Cloud paradigms and practices for computational and data-enabled science and engineering. Computing in Science Engineering **15**(4), 10–18 (July 2013)
28. Rajan, D., Thain, D.: Designing self-tuning split-map-merge applications for high cost-efficiency in the cloud. IEEE Transactions on Cloud Computing **5**(2), 303–316 (April 2017)
29. Rajan, D., Thrasher, A., Abdul-Wahid, B., Izaguirre, J.A., Emrich, S., Thain, D.: Case studies in designing elastic applications. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. pp. 466–473 (May 2013)
30. Rajan, D., Canino, A., Izaguirre, J.A., Thain, D.: Converting a high performance application to an elastic cloud application. In: IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom). pp. 383–390. IEEE (2011)
31. Raveendran, A., Bicer, T., Agrawal, G.: A framework for elastic execution of existing mpi programs. In: 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum. pp. 940–947 (May 2011)
32. Rodrigues, V.F., da Rosa Righi, R., da Costa, C.A., Singh, D., Munoz, V.M., Chang, V.: Towards combining reactive and proactive cloud elasticity on running

hpc applications. In: Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS. pp. 261–268. INSTICC, SciTePress (2018)

33. Roloff, E., Diener, M., Carissimi, A., Navaux, P.O.A.: High performance computing in the cloud: Deployment, performance and cost efficiency. In: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings. pp. 371–378 (Dec 2012)
34. da Rosa Righi, R., Rodrigues, V.F., da Costa, C.A., Galante, G., de Bona, L.C.E., Ferreto, T.: Autoelastic: Automatic resource elasticity for high performance applications in the cloud. IEEE Transactions on Cloud Computing **4**(1), 6–19 (Jan 2016)
35. da Rosa Righi, R., Rodrigues, V.F., da Costa, C.A., Kreutz, D., Heiss, H.U.: Towards cloud-based asynchronous elasticity for iterative HPC applications. Journal of Physics: Conference Series **649**, 012006 (oct 2015)
36. da Rosa Righi, R., Rodrigues, V.F., Rostirolla, G., da Costa, C.A., Roloff, E., Navaux, P.O.A.: A lightweight plug-and-play elasticity service for self-organizing resource provisioning on parallel applications. Future Generation Computer Systems **78**, 176 – 190 (2018)
37. Sun, Y., Wang, C.L.: Solving irregularly structured problems based on distributed object model. Parallel Computing **29**(11-12), 1539–1562 (Nov 2003)
38. Vecchiola, C., Pandey, S., Buyya, R.: High-performance cloud computing: A view of scientific applications. In: 10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN). pp. 4–16. IEEE (2009)
39. Vu, T.T., Derbel, B.: Link-heterogeneous work stealing. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. pp. 354–363 (May 2014)
40. Webster, J., Watson, R.T.: Analyzing the past to prepare for the future: Writing a literature review. MIS Quarterly **26**(2), xiii–xxiii (2002)
41. Yang, X., Wallom, D., Waddington, S., Wang, J., Shaon, A., Matthews, B., Wilson, M., Guo, Y., Guo, L., Blower, J.D., Vasilakos, A.V., Liu, K., Kershaw, P.: Cloud computing in e-science: research challenges and opportunities. The Journal of Supercomputing **70**(1), 408–464 (Oct 2014)
42. Zhai, Y., Liu, M., Zhai, J., Ma, X., Chen, W.: Cloud versus in-house cluster: Evaluating amazon cluster compute instances for running mpi applications. In: SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–10 (Nov 2011)
43. Zhang, J., Lu, X., Panda, D.K.: Performance characterization of hypervisor-and container-based virtualization for hpc on sr-iov enabled infiniband clusters. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 1777–1784 (May 2016)
44. Zhang, J., Lu, X., Panda, D.K.D.: Designing locality and numa aware mpi runtime for nested virtualization based hpc cloud with sr-iov enabled infiniband. In: Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. pp. 187–200. VEE '17, ACM, New York, NY, USA (2017)
45. Zimmermann, O.: Architectural refactoring for the cloud: a decision-centric view on cloud migration. Computing **99**(2), 129–145 (Feb 2017)

Towards a Platform for Sharing Quantum Software

Frank Leymann [0000-0002-9123-259X], Johanna Barzen [0000-0001-8397-7973] and Michael Falkenthal [0000-0001-7802-1395]

Institute of Architecture of Application Systems, University Stuttgart, Germany
`Lastname@informatik.uni-stuttgart.de`

Abstract. Quantum computers solving real-world problems are expected to become general available within the next few years. But software for quantum computers require very different skills compared to creating software for traditional computers or networks. Thus, a community-driven approach to creating software for quantum computers will foster a wide-spread use of this innovative technology. Also, a platform for quantum software may provide a business model for several user groups.

Keywords: Quantum computing, software engineering, middleware, platforms, cloud computing.

1 Introduction

Quantum computing is becoming a reality: the first quantum computers are already commercially available or are about to be launched [5], [3], [4], [7]. The time at which quantum computers will solve problems that traditional computers can practically no longer solve (so-called "Quantum Supremacy" [10]) can be expected in the next few years [9]. Even earlier, non-ideal quantum computers can be used in practice [11].

Creating algorithms or software for quantum computers is significantly different from todays practice. Efforts in establishing a discipline of quantum software are significantly underrepresented [1]. Although there are a large number of algorithms for quantum computers (e.g. on websites like [6], in textbooks like [8], in scientific publications like [1]), which algorithm can be used in which situation requires a comprehensive understanding of the theory and technology, which users typically do not have. Even if a suitable algorithm is found, its conversion into an executable program requires deep knowledge of the environment of the respective quantum computer.

2 Quantum Software Platform

This is where the concept of the proposed Quantum Software Platform (QuSP) comes in. Components of the QuSP and user groups involved are shown in Fig. 1.

2.1 Overview

The algorithms for the QuSP come from many different sources (A) (NB: letters and numbers refer to labels in Fig. 1) such as the web, published articles or books. These algorithms are stored in a special database, the quantum algorithm catalog (1).

A public community (analogous to an open source community) or specialists of the platform operator (B) can access this special database and analyze, clean and unify the algorithms (2). As a result, each quality-assured algorithm is stored in the Quantum Algorithm Repository (3).

Based on the quality-assured algorithms, developers (C) can now implement these algorithms for execution on a quantum computer (4). These programs are also quality-assured (5) and stored in a quantum program repository (6).

End users (D) of the platform can now search for quality-assured algorithms and programs in the QuSP (7). If an algorithm for a certain problem is not found or if an algorithm is not implemented by a program, the end user can make corresponding requests (7) to the community. For delivery, an algorithm or program is packaged (9)

2.2 Population

Algorithms to be considered are captured by metadata. E.g. its reference is essential so that it can be retrieved and viewed. If it is available online, this reference is the corresponding link, otherwise the exact literature reference is given. It is also described which problems the algorithm is claimed to solve, initial information about the properties of the algorithm etc. is given.

Candidates for the quantum algorithm catalog can be identified in different ways. For example, community members can capture algorithms, or crawlers may automatically detect relevant publications.

2.3 Preparation

Next, the community agrees on the maturity of an algorithm [12]. If mature, it will be analyzed, unified and stored in the quantum algorithm repository. Analysis determines which problems the algorithm actually solves and with which properties: e.g. statements are made about the acceleration an algorithm achieves compared to certain classical algorithms [13], with how many qubits an algorithm is already useful [9], etc. During unification, the algorithm is brought into a common format (e.g., represented as a software pattern). If not mature, the algorithm is cleansed, i.e. its unsuitability is captured and the catalog is annotated accordingly so that this algorithm is not considered again.

These tasks can be performed by different user groups, e.g. a public community (analogous to an open source community), or specialists of the platform operator. For this purpose, the platform contains corresponding collaboration tools [12].

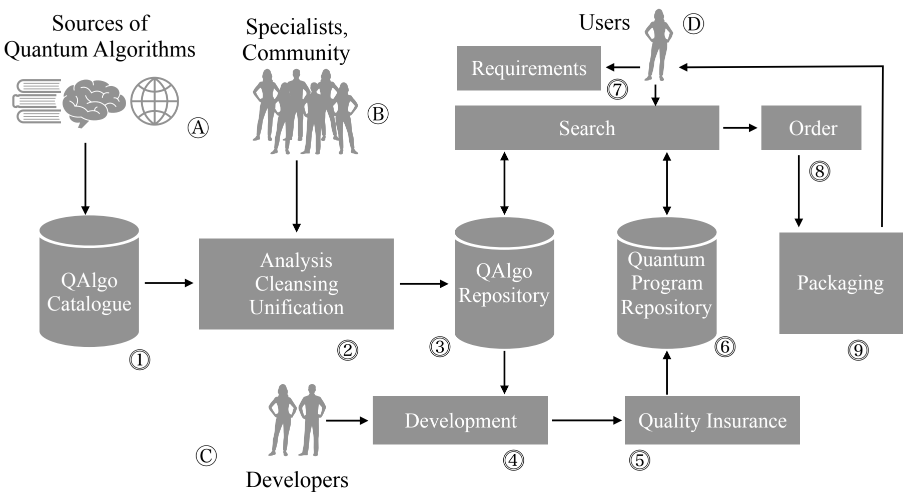

Figure 1 - Architecture of the quantum software platform QuSP.

2.4 Program Development

Quantum algorithms that meet specific requirements (such as the number of qubits required, fit for a hardware architecture) may be implemented so that they can run on the quantum computer of certain vendors. These quantum programs are examined for their quality (functional tests, etc.) before they are transferred to the quantum program repository. This can be done by an open source community. Also, specialists of the manufacturer of a certain quantum computer can implement those algorithms.

2.5 Usage

Customers of the QuSP can search for both, quantum algorithms and quantum programs. If an algorithm for a problem is not found, requirements to provide the appropriate algorithms can be imposed. If an algorithm is found but no corresponding implementation as a quantum program is found, an implementation can be requested. Providing algorithms and programs on request may be charged. If a quantum program is retrieved, the program is packaged so that it can be provisioned into its target environment.

3 Roles Involved

3.1 Platform Operators

The platform may be operated as a business, e.g. a fee can be charged for (successful) searches, for access to the unified representation of the algorithm etc. For programs that implement an algorithm, an even higher price can be charged. Satisfying requirements have to be paid.

3.2 Users

The search of different sources for suitable quantum algorithms as well as the assessment of their suitability is not only very time-consuming, but also requires highly specialized personnel. The use of the QuSP thus represents a significant advantage for users and often makes the use of quantum computers possible in the first place.

The same applies to the creation of a quantum program: the environment for developing such a program must be understood, and it is often even specific to a certain hardware of a supplier. Likewise, quantum algorithms are usually formulated independently of a specific hardware, so that adaptations to an appropriate target platform are necessary [14]. The purchase of a program that implements an algorithm for a specific quantum computer thus represents a considerable savings potential for a user.

3.3 Software Vendors

Quantum programs can be created by software companies and offered in the platform. Requirements that customers place on quantum programs and their target environments can thus be met by software companies. Software companies can specialize in different hardware platforms or development environments of quantum computers and thus achieve competitive advantages.

3.4 Hardware Vendors

Hardware vendors can also offer quantum programs: these programs are optimized for the hardware which becomes more attractive, of the programs must be paid.

3.5 Consulting Companies

Consulting companies are often specialized in industries (pharmaceuticals, finance...) or cross-sectional topics (optimization, simulation...) that can benefit from quantum computing. Personnel with knowledge in the field of quantum computing is rare, i.e. consulting firms can benefit in particular from the quality-assured algorithms of the QuSP in order to incorporate quantum technologies into their specialized consulting services. Users thus have access to corresponding consulting services.

4 Conclusion and Outlook

The sketched platform for quantum software will enable a much broader group of people and companies to take advantage of the benefits of quantum computing. Initial steps to create a prototype of such a platform is underway at our institute.

References

1. Ambainis, A., et. al.: “Quantum Software Manifesto“. (2017), http://www.qusoft.org/wp-content/uploads/2018/02/Quantum-Software-Manifesto.pdf
2. Coles, P.J., et al: „Quantum Algorithm Implementations for Beginners“. arXiv:1804.03719v1 (2018).
3. IBM: “Quantum Devices and Simulators”. (2018). https://www.research.ibm.com/ibm-q/technology/devices/#ibmq-20-tokyo
4. Intel: “Intel’s 49-Qubit quantum processor”. (2018). https://newsroom.intel.com/wp-content/uploads/sites/11/2018/05/49-qubit-processor-tangle-lake-infographic.jpg
5. Jelly, K.: “A preview of Bristlecone, Google’s new quantum processor”. Google (2018). https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html
6. Jordan, S.: "Quantum Algorithm Zoo“. (2018). https://math.nist.gov/quantum/zoo/
7. Linn, A.: “With new Microsoft breakthroughs, general purpose quantum computing moves closer to reality”. (2017). https://news.microsoft.com/features/new-microsoft-breakthroughs-general-purpose-quantum-computing-moves-closer-reality/
8. Lipton, R.J., Regan, K.W.: „Quantum Algorithms via Linear Algebra“. MIT Press 2014.
9. Mohseni, M., et al.: “Commercialize early quantum technologies“. Nature Vol. 543, March 2017.
10. Preskill, J., in „The Theory of the Quantum World“ (eds Gross, D., Henneaux, M. & Sevrin, A.) 63–80 (World Scientific, 2011).
11. Preskill, J.: “Quantum Computing in the NISQ era and beyond”. Quantum 2, 79 (2018).
12. Reiners, R.: „An Evolving Pattern Library for Collaborative Project Documentation“. Dissertation RWTH Aachen, 2013.
13. Rønnow, T. F., et al.: „Defining and detecting quantum speedup“. Science Vol. 345(6195) July 2014.
14. Tannu, S.S., Qureshi, M.K.: “Not All Qubits Are Created Equal - A Case for Variability-Aware Policies for NISQ-Era Quantum Computers”. arXiv:1805.10224 (2018).

(All links have been followed on June 2019, 2018).

How to Reconstruct Musical Experiences from Historical Texts: Methodological Issues

Claes Neuefeind[1] and Brigitte Mathiak[2] and Frank Hentschel[3]

[1] University of Cologne, Cologne Center for eHumanities, 50923 Köln, Germany
[2] University of Cologne, Institute for Digital Humanities, 50923 Köln, Germany
[3] University of Cologne, Department of Musicology, 50923 Köln, Germany
{c.neuefeind,bmathiak,fhentsch}@uni-koeln.de

Abstract. In this paper, we outline an approach to reconstruct musical experiences from historical texts that makes use of techniques from the field of Digital Humanities (DH) to support musicological research on musical expressivity. Based on the assumption that the systematic investigation of textual evidence of contemporary musical experiences can give insights in the relation between compositional practice and specific expressive qualities, we sketch a digitally enhanced workflow to access these experiences. The resulting tool allows to search for expressive qualities in a corpus of 19th-century music periodicals and journals to determine which works are mentioned most frequently in those contexts. This forms the basis for an in-depth musicological analysis of the characteristics of these works. In this paper, we focus on methodological issues that arise from combining qualitative musicological analysis with quantitatively oriented digital methods from the field of DH, where the ideas and methodological issues discussed in this paper are part of a broader agenda in musicology that can be coined as *historical music psychology*.

Keywords: Musical expressivity, 19th-century music, Digital Humanities.

1 Introduction

The emotional impact of music is a well known fact. When we listen to music, we recognize or feel emotions or moods. Many studies try to clarify why and how music elicits or communicates affect; most of them base their research on test persons (see for overviews [1], [2], [3]). But what about the audiences and listeners in former times, say, in the 19th century? How did people actually experience music in its historical context? Did people at that time experience music in the same way as we do today? Which compositional patterns triggered these experiences? Based on these questions, we outline a new methodological approach in musicology in this paper, which is part of a broader agenda that can be coined as *historical music psychology*.

The key idea of this approach is that access to contemporary musical experiences can shed some light on the relation between compositional practice and expressive qualities. To be sure, more recent research has made clear that acoustic features that cannot be deduced from the score, as well as the performance also influence the way music affects listeners (e.g. [4], [5]). However, since we have no access to 19th-century performances or recordings we focus on compositional features. With respect to the listening experience of the 19th century, recensions and articles in music-related magazines and periodicals are the most important sources. To give an example of textual evidence where the emotional quality of music is explicitly described, we refer to [6], where among others the following excerpts are described:

Felix Mendelssohn, Symphony No. 3, *Scottish* (1803)
Anonymous. "The Birmingham Musical Festival." *The Musical World*, vol. 24, no. 36 (September 8, 1849), pp. 564–9, here p. 568: "The concert began with the **Third symphony** of Mendelssohn - that in **A minor**. […] the fire and impetuosity imparted to the whole of that movement, and the magnificent ensemble of the hymn of thanksgiving with which the symphony so nobly concludes, were such as we have rarely heard, even in London […]."

Pyotr Ilyich Tchaikovsky, Symphony No. 5 (1888)
Rosa Newmarch. *Tchaikovsky. His Life and Works*. London: William Reeves, 1908, p. 99: "The introduction to the *Finale – Andante maestoso –* is penetrated with religious feeling. […] The *Finale* itself (*Allegro vivace*) grows gradually clearer as it proceeds, as though the heart had cast off a load of suffering and God's world shone out bright once more."

As the examples show, there are many different ways to describe musical experiences. These sum up to a vocabulary of expressive qualities – which in turn are used to describe musical experiences. In order to make reliable statements about how a certain music was experienced by its contemporaries, as many sources as possible must be tracked down. Therefore, a systematic search for such sources is required. Thus, the question arises how descriptions and work mentions can be formally related to each other so that they can be systematically accessed.

To reconstruct musical experiences from historical texts we need to find textual evidence that can be used as a basis for an in-depth musicological analysis. This implies that these sources do not only use descriptors of musical expression but that they do so with respect to specific works or, better, work excerpts. We therefore need to develop a procedure that allows on the one hand to search for particular expressive qualities (e.g. 'sad' 'majestic' or 'noble') in a corpus of 19th-century music periodicals and journals and on the other hand to determine which works are mentioned most frequently in those contexts. On this basis, the characteristics of the works in question can then be analyzed for similarities by a musicologist.

2 Data

As stated above, the only way to access contemporary musical experiences is through text. Thus the whole idea ultimately depends on a reliable textual basis that contains as much material as possible. In previous research ([6], [7]), the *Retrospective Index to Music Periodicals* (RIPM, [8]) was used, which to date is by far the largest collection of music-related periodicals. By the time of writing, the RIPM database includes about 388 titles from more than 20 different countries in Europe and the Americas, out of which 299 provide access to the full texts, and covering the time span between 1760 and 1966. In total, the corpus includes approximately 930,000 annotated entries or 1.18 mio pages of text. In addition, based on preliminary work within the scope of the DFG-funded project “Musikalische Preisausschreiben” [9], this corpus may be further extended by other freely available music periodicals and magazines.

3 Methodology

Our starting point is the methodology developed in [7], where the following procedure is conducted to select historical source texts to find textual evidence for the reception of music as being ‚uncanny‘ and to subsequently analyze the works described in these texts:

1. Define a ‘semantic field’
2. Search for all word forms of this field in the RIPM database
3. Manual filtering based on direct contexts (skip obviously irrelevant ones)
4. Select “suitable documents” (containing references to works/pieces)
5. Musicological analysis of the works mentioned in the documents

The question we want to raise here is: how can we do better? To this end, we take on a “DH perspective,” where our main concern is which parts of this procedure can be automatized or at least be supported by computational techniques. Except for the steps 3 and 5, which are explicitly manual operations, we see a thorough potential to support the musicologist’s work by automatization. Having said that, we can derive the following steps for a DH-supported procedure:

1. Generate exemplary **semantic fields** and refine them manually
2. Recognize/annotate **work mentions** (relevant documents)
3. **Search** for common contexts of semantic fields and work mentions

These three steps are the prerequisite for an in-depth analysis of the search results and the musicological analysis of compositions referred to in the texts (extracting compositional features, defining expression types etc.). The rest of this chapter is organized alongside these three steps: First, we describe an approach to automatically generate semantic fields, then we discuss approaches for the identification of work mentions, and finally we sketch the basic design of an according search functionality.

3.1 Semantic fields

First of all, when we talk about *semantic fields* in this context, we do not refer to a specific linguistic concept (like e.g. [10]). Instead, it is a more or less preliminary term to describe an unordered set of thematically related words. In its simplest form, a semantic field can be seen as a manually assembled list of words supposed to intersect in some central aspects of their meaning or to refer to closely related phenomena, as in the following example for ‘uncanny’, which is taken from [7]:

Erschauern, geisterhaft, gespenstig, gespenstisch, Grauen, grauenerregend, grauenhaft, grauenvoll, Graus, Grausen, grausig, gruselig, horribel, mysteriös, schauderhaft, Schauer, schauerlich, schauervoll, schaurig, unheimlich; abominable, demoniacal, demonic, dismay, dread, dreadful, eerie, eery, ghastly, ghostly, gruesome, horrible, horrid, horrific, mysterious, scary, shiver, shudder, spookish, spooky, uneasy, uncanny; angoissant, épouvante, frémir, frisson, horreur, horrible, inquiétant, lugubre, sinistre, téné breux; misterioso, orrendo, orribile, orridezza, orrore, sinistro, spaventoso, mistico, inquietante.

In this example, the selection of terms is actually based on the personal language competence and hermeneutic experience of an individual historian using manually selected contemporary texts and contemporary dictionaries, encyclopedias, etc. from different languages (e.g. Grimm's *Deutsches Wörterbuch* [11] or *Dornseiff* [12] for German). Such an approach is necessarily restricted since the amount of texts an individual researcher can survey is limited and possibly biased. Another way of assembling related terms is to rely on resources dedicated to (in our case) emotional vocabulary like *NRC Emolex* [13] or to extract the according parts from *WordNet* [14] and then adapt them to contemporary orthography, for these resources are not based on historical (19th-century) language usage.

As an alternative, to introduce a fully automated approach on the basis of contemporary sources, we propose to use word embeddings to find sets of similar words. Besides count-based approaches such as *SVD-ppmi* [15], the most popular approaches to compute word embeddings are *Word2Vec* [16] and *GloVe* [17], both relying on feature learning techniques based on neuronal networks. The operationalization can be outlined as follows: first we compute word embeddings based on a raw text corpus of as much 19th-century language we can find (using additional contemporary material such as *Deutsches Textarchiv* [18]). Then we query these embeddings for similar words. However, in this context "similar" simply means that a word may replace each other, so that "good" is close to "bad". As a consequence, we need additional manual input. This means that beyond automatization, there still is a great need for manual selection and refinement. This can again be done with the help of contemporary dictionaries and encyclopedias, as well as with the help of corpus-based samples.

3.2 Detecting Work Mentions

As for the detection of work mentions, we return to the examples from above:

Felix Mendelssohn, Symphony No. 3, *Scottish* (1803)

Anonymous. "The Birmingham Musical Festival." *The Musical World*, vol. 24, no. 36 (September 8, 1849), pp. 564–9, here p. 568: "The concert began with the **Third symphony** of Mendelssohn - that in **A minor**. […] the fire and impetuosity imparted to the whole of that movement, and the magnificent ensemble of the hymn of thanksgiving with which the symphony so nobly concludes, were such as we have rarely heard, even in London […]."

Pyotr Ilyich Tchaikovsky, Symphony No. 5 (1888)

Rosa Newmarch. *Tchaikovsky. His Life and Works*. London: William Reeves, 1908, p. 99: "The introduction to the *Finale – Andante maestoso* – is penetrated with religious feeling. […] The *Finale* itself (*Allegro vivace*) grows gradually clearer as it proceeds, as though the heart had cast off a load of suffering and God's world shone out bright once more."

Even though the name *Scottish* has not been used by the author the explicit numbering and the mentioning of the key make the identification fairly easy. In contrast to this, the second example is not at all easy. Since the references are highly ambiguous, it can only be attributed by collecting clues from a number of sources and connecting them to make an educated guess.

For this task, we can combine three different approaches to the task of entity recognition: knowledge-driven (lists of names and pieces), pattern induction (based on linguistic patterns, [19]), or based on machine learning techniques such as conditional random fields (CRF, [20]), or – again – by using word embeddings. Musicologist and linguist will have to work hand in hand, starting with the knowledge-driven approach and refining it manually, which in turn provides training data for the more sophisticated machine learning methods. Both patterns and CRFs work very well with sparse training data and for the word embeddings we can again use the much larger data set representing the language in question that was used in the task of generating semantic fields.

3.3 Semantic Search

As a final step, a search functionality has to be provided that allows to find all work mentions occurring within the context of specific semantic fields. The basic idea is to make use of the concept of query expansion, where every search term triggers a search for all words that are members of the same semantic field. This can be realized with established search technologies like e.g. elasticsearch [21], where the pre-computed sets of words (i.e. the semantic fields) can be fed into the system as so-called 'synonym tokens' via an external configuration file. A single search term will then return all texts that contain any of the elements of the specific semantic field the search term belongs to.

Since this will increase the recall significantly, a proper ranking of the search results is crucial. Hereby, the annotations of work mentions can be used to filter and rank the search results. Weights can be determined by the occurrence and position of work mentions, so that results in which such annotations occur close(r) to search terms can be prioritized. To avoid the loss of potentially misclassified work mentions (e.g. in case work mentions weren't properly annotated), nothing is sorted out but only placed at the end of the result list – which in turn calls for a thorough manual inspection by a musicologist.

4 Musicological Analysis

With the three steps described above, we have a computationally supported methodology at hand that helps to define a subset of relevant documents from a large corpus of music-related texts from the 19th century. On this basis, an in-depth analysis can be conducted. The musicologist will not only check whether the results do actually contain relevant descriptors of specific compositions but also whether they refer to specific passages within the compositions. The more specific the reference is the more important will be the source. A qualitative text analysis will help to structure the sources. The excerpts to which the sources refer can then be analyzed with respect to common compositional features. We expect to find certain features that recur in many compositions and allow hypotheses or even conclusions about what musical features have typically been used to produce a certain expressive effect (cf. [7]). Moreover, we expect to find expression types, i.e. certain combinations of features that are regularly used to communicate certain expressive qualities [22]. In the long run, we hope to contribute to the reconstruction of the expressive vocabulary of 19th-century music.

5 Summary and Outlook

In this paper, we outlined a digitally enhanced workflow to reconstruct musical experiences from historical texts. After the generation (and manual refinement) of semantic fields and the identification and annotation of work mentions in a corpus of 19th-century music periodicals, journals and other publications, a corpus search for expressive qualities returns a list of work mentions that are subject to an in-depth musicological analysis. Among the benefits of such a digitally enhanced workflow are better and faster access to the data and the possibility to quantify the results.

A potential extension of this approach could be a kind of ‘heat map’ visualization to support the selection of ‘good’ semantic fields. The extension is inspired by ideas from *Culturomics* [23] and *Distant Reading* [24], aiming at the reconstruction of the historical vocabulary of a defined timespan by approximating a significant number of semantic fields. On this basis, the occurrence and distribution of these fields over time and space could be visualized. This would allow to investigate the temporal and geographical distribution of terms that are relevant for the description of musical perception, where timelines and graphics are intended to visualize changes and distributions of semantic fields across 19th-century Europe.

As stated in the beginning, the ideas and methodological issues discussed in this paper are part of a broader agenda in musicology that can be coined as *historical music psychology*. By designing a systematic, computationally supported workflow to reconstruct musical experiences from historical texts, we hope to get better insights in the relation between compositional patterns and specific expressive qualities - and thereby contribute to this agenda.

References

1. Gabrielsson, A., Juslin, P. N.: Emotional Expression in Music. In: Davidson, R.J. (ed.): Handbook of affective sciences, pp. 503–534. (2003). Oxford, New York u.a.: Oxford Univ. Press (2003).

2. Thompson, W. F.: Music and Emotion (pp. 169–206). In: Thompson, W. F.: Music, Thought, and Feeling: Understanding the Psychology of Music (2 ed.). Oxford: Oxford University Press (2015).
3. Margulis, E. H.: The Psychology of Music: A Very Short Introduction. Oxford: Oxford University Press (2019).
4. Lange, E. B., Frieler, K.: Challenges and Opportunities of Predicting Musical Emotions with Perceptual and Automatized Features. In: MUSIC PERCEPT 36 (2), 217–242 (2018). DOI: 10.1525/MP.2018.36.2.217.
5. Juslin, P. N.: Emotion in Music Performance. In: Oxford Handbook of Music Psychology (1 ed.), Edited by Susan Hallam, Ian Cross, and Michael Thaut, Print Publication Date: Dec 2008, Online Publication Date: Sep 2012, DOI: 10.1093/oxfordhb/9780199298457.013.0035.
6. Hentschel, F.: Expression Types of 19th-Century Symphonic Music: The Cases of the Glorifying Hymnic and the Majestic Chorale. Open Science Framework (OSF Preprints), 2018 (DOI: 10.31219/osf.io/hgqnt).
7. Hentschel, F.: Musik und das Unheimliche im 19. Jahrhundert. In: *Archiv für Musikwissenschaft* 73 (1), pp. 9–50 (2016).
8. RIPM homepage, https://www.ripm.org/, last accessed 2019/06/19.
9. Musikalische Preisausschreiben, https://gepris.dfg.de/gepris/projekt/273448812, last accessed 2019/06/19.
10. Johnson-Laird, P. N.; Oatley, K.: The language of emotions: An analysis of a semantic field. In: *Cognition & Emotion* 3 (2), 81–123 (1989). DOI: 10.1080/02699938908408075.
11. Grimm, J. and W.: Deutsches Wörterbuch. 16 volumes. Leipzig: S. Hirzel (1854-1961).
12. Dornseiff, F.: Der deutsche Wortschatz nach Sachgruppen. Berlin, Leipzig: de Gruyter (1934).
13. NRC Emolex, http://www.saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm, last accessed 2019/06/19.
14. WordNet, https://wordnet.princeton.edu/, last accessed 2019/06/19.
15. Levy, O., Goldberg, Y., Dagan, I.: Improving Distributional Similarity with Lessons Learned from Word Embeddings. In: Transactions of the Association for Computational Linguistics, v. 3, 211-225 (2015).
16. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient Estimation of Word Representations in Vector Space (2013). http://arxiv.org/pdf/1301.3781v3, last accessed 2019/06/19.
17. Pennington, J., Socher, R., Manning, C.: GloVe: Global Vectors for Word Representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar. Stroudsburg, PA: Association for Computational Linguistics, 1532–1543 (2014).
18. Deutsches Textarchiv (DTA), http://www.deutschestextarchiv.de/, last accessed 2019/06/19.
19. Boland, K., Ritze, D., Eckert, K., Mathiak, B.: Identifying References to Datasets in Publications. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C. (ed.): Theory and Practice of Digital Libraries (7489). Berlin, Heidelberg: Springer (Lecture Notes in Computer Science), pp. 150–161 (2012).
20. Lafferty, J.; McCallum, A., Pereira, F. C. N.: Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In: Brodley, C. E. (ed.): Machine learning. Proceedings of the eighteenth international conference. San Francisco, CA: Kaufmann, pp. 282–289 (2001).
21. Elasticsearch homepage, https://www.elastic.co, last accessed 2019/06/19.
22. Barzen, J., Breitenbücher, U., Eusterbrock, L., Falkenthal, M., Hentschel, F., Leymann, F.: The vision for MUSE4Music. In: *Computer Science. Research and Development* 32 (3-4), pp. 323–328 (2017). DOI: 10.1007/s00450-016-0336-1.

23. Michel, J.-B., Shen, Y. K., Aiden, A. P., Veres, A., Gray, M. K.; Pickett, J. P.: Quantitative analysis of culture using millions of digitized books. In: *Science* 331 (6014), 176–182 (2011). DOI: 10.1126/science.1199644.
24. Moretti, F.: Graphs, maps, trees. Abstract models for a literary history. London: Verso (2005).

CO_2-efficient Home Energy Management: A Service-Oriented Approach [⋆]

Laura Fiorini[1]

University of Groningen, Department of Distributed Systems, Nijenborgh 9, 9747 AG, Groningen, The Netherlands
`l.fiorini@rug.nl`

Abstract

Increasing concerns on climate change have brought the attention to the environmental impact of building energy consumptions; in Europe, residential and commercial building are attributed 36% of CO_2 emissions. Increasing diffusion and use of appliances have significantly raised the households' need for electricity, and controlling the residential energy use has become an important aspect of the power grid management. With multiple energy sources available, the complexity of households as energy systems increases, while growing energy consumptions imply a higher environmental impact that, in the context of decarbonization, needs to be addressed. Developments in automation are pivotal in the transition towards multi-energy systems and smart buildings, where the integration of diverse energy carriers, such as electricity and gas, has shown potential economic and energetic benefits. A Home Energy Management System (HEMS) aims at monitoring and controlling energy consumption and production of a household, while scheduling the use of available resources. A Service-Oriented Architecture can simplify the design and management of smart homes by allowing components to dynamically find each other and interoperate. Standardized interfaces and messages allow the home management system to integrate new components and services as they become available. Among these, Smart Meters are a key component of energy management systems. These provide for real-time information and take decisions related to energy consumptions according to the data provided by other available services, for instance, the output of the solar panel service, or the CO_2 emission factor provided by the electricity supplier service.

Within the energy transition context, the main research questions of this work focus on the integration of multiple energy carriers within household energy systems, how to model them, and how to promote a sustainable management. We investigate to what extent the environmental impact of households can be reduced by integrating multiple energy sources, namely, electricity, natural gas,

[⋆] Supported by the Netherlands Organization for Scientific Research under the NWO MERGE project, contract no.647.002.006; and by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no.734599.

and solar generation, taking advantage of variations over time of CO_2 intensity of different energy carriers.

The Smart Home model we propose can be realized as a Service-Oriented Architecture and designed following Service-Oriented Computing principles. Depending on the household size and the season, a Smart Home may have up to six smart appliance services, a heating and cooling service, and a domestic hot water service. The energy behavior of each appliance service is modeled as a series of tasks requiring different levels of electric and/or thermal power. The appliance's status and energy consumptions are controlled by a Smart Meter. The heating and DHW services are provided by multiple technologies, namely a gas-fired boiler, an electric heat pump, an electric water heater, and a thermal store. Depending on the service and the user's preferences, we define multiple possible levels of flexibility with respect to allowed delays of individual appliance service, temperature range of thermal services, and energy sources. We also include a local energy production service provided by a system of photovoltaic panels and an electricity storage service provided by a battery.

Such a Smart Home is managed by a coordinator service, called HEMS, which determines the best operation scheduling of the other services and technologies according to users' preferences, information coming from the energy supplier services, e.g., CO_2 emission or price signals, and from a weather information service, e.g., outside temperature and sky conditions. To deal with information uncertainty, the coordinator service adjusts the scheduling for the following 6 hours every half a hour by means of Model Predictive Control.

Preliminary results show that the proposed HEMS can significantly reduce CO_2 emissions, while satisfying users' comfort preferences in terms of temperature and allowed delay of main household appliance services. Considering configurations with increasing levels of load flexibility and integration of energy carriers, emission savings vary between 2% and 27%, compared to the reference case with low flexibility and traditional operation modes. The presence of more technologies allows for the improvement of supply's efficiency for space heating and domestic hot water demand, maintaining the total energy consumptions mostly unchanged. Further emission and energy consumption savings can be achieved with the use of batteries coupled with solar panels.

Further works will focus on investigating different models for assessing the CO_2 intensity of electricity, namely, mix and marginal methods.

References

1. L. Fiorini and M. Aiello, "Household CO_2-efficient energy management," *Energy Informatics*, vol. 1, no. Suppl 1, pp. 21–34, 2018.
2. L. Fiorini and M. Aiello, "Predictive CO_2-efficient Scheduling of Hybrid Electric and Thermal Loads," In: Proceedings of 3rd IEEE International Conference on Energy Internet, pp. 392–297, 2019.
3. L. Fiorini and M. Aiello, "Energy Management for User's Thermal and Power Needs: A Survey," Submitted, 2019.

4. M. Aiello, “The Role of Web Services at Home.”, In: IEEE Web Services-based Systems and Applications (WEBSA at ICIW) IEEE Computer, 2006.

Optimising Local Energy Storage for Smart Grid Connected Offices

Brian Setz$^{1[0000-0002-9750-2888]}$

Department for Service Computing, Institute of Architecture of Application Systems, University of Stuttgart
brian.setz@iaas.uni-stuttgart.de

The smart grid promises to transform the current electrical grid to one that functions more cooperatively, responsively, and organically [4]. Energy consumers in a future smart grid can become energy providers, delivering energy back to the grid. Consumers have the potential to select which energy provider to purchase electricity from; perhaps a neighbouring building is producing more power than they consume, and thus sell it to the grid for a favourable price. As a result, there are many energy providers, selling and purchasing energy at different prices. Additionally, the availability of local energy storage enables consumers and producers to store energy when it is most beneficial for them. This leads to an interesting optimization problem.

In this work we investigate the effect that local energy storage has on optimizing the costs for smart offices connected to the smart grid, by scheduling the operation of devices in an optimal manner using a service-based approach. Optimal refers to minimizing the energy cost to maximize financial savings. Policies can be used to describe the behaviour of these devices, determining when and how they should be operated [1]. For example, a fridge needs to operate at periodic intervals to maintain its temperature, and a laptop charger needs to operate a total amount of time to fully charge a laptop. We define a scheduling problem that has as input the available energy providers, the devices to operate and their policies, the locally produced renewable energy, and the local energy storage. The output is an optimal device schedule with 15-minute timeslots.

The energy prices are collected from the REST API of the Amsterdam Power Exchange (APX) Group, an energy exchange operating the spot markets for electricity in the Netherlands. The energy providers are generated by sampling a normal distribution with a mean value equal to the APX price, and a small standard deviation. Each energy provided also has a limited amount of power it can provide, this is to simulate a smart grid where agents are able to deliver small amounts of power to the grid, for example when their solar panels are overproducing. The amount of power that an energy provider can deliver is sampled from a uniform distribution with a fixed upper and lower bound.

Two additional energy providers are added for locally generated renewable energy, one for solar panels and one for wind turbines. To define the wind turbine provider we use a model that determines the power generation based on the wind speed, cutin wind speed, survival wind speed, and the power curve of a particular wind turbine [2]. For the solar panels provider, we use a model that is trained using data from solar panels located in the Netherlands, where the output of the solar panels is correlated to the cloud coverage. We include a local energy storage model based on the Tesla Power Wall as part of the scheduling problem.

We solve the scheduling problem using a uniform-cost search optimization algorithm. This algorithm works by first generating all partial solutions for the first time slot [3]. These partial solutions are stored in a priority queue; the partial solution with the lowest cost is at the top of the queue. Next, the first partial solution in the queue is dequeued, the partial solutions for the next time slot are calculated and stored in the queue. Every time, the partial solution with the lowest cost is expanded further. Cost is defined as either the total cost of the consumed energy. In each time slot it is possible to operate the devices, and change the state of the local storage from charging to discharging, and vice-versa.

The number of devices and timeslots determines the complexity of the search space. We apply two techniques to limit the search space. First, we ensure that the partial solutions that are generated are valid solutions. A solution is valid when the device policies are not violated. For example, if a device is scheduled more often than a policy demands, the policy is violated. If a policy is violated, the partial solution is removed. Second, we check that there are no duplicate partial solutions. A duplicate solution has the same cost after the same number of timeslots have been expanded, and reaches the same state after execution.

The architecture of the system is as follows: the *Database* stores the relevant policies for each device, as well as the energy prices for each energy provider. The energy data is updated by the *Extractor Service*, which is responsible for requesting the latest data from externally located API's. The *Coordinator Service* queries the database for the latest information and feeds this data to the *Scheduler Service*. The scheduler generates a schedule based on the inputs and returns this schedule to the coordinator. Next, the coordinator translates the schedule into commands that are understood by the *IoT Gateway*.

Our preliminary results show that, given the same smart office optimization infrastructure to react to energy prices, introducing local energy storage allows us to decrease the energy costs even further. The reasons for the decreased cost are twofold. First of all, when local renewables overproduce, the excess energy can be stored for later use. And second, when energy prices are low, the energy can be purchased and stored at this low price.

References

1. Georgievski, I., Degeler, V., Pagani, G.A., Nguyen, T.A., Lazovik, A., Aiello, M.: Optimizing energy costs for offices connected to the smart grid. IEEE Transactions on Smart Grid **3**(4), 2273–2285 (Dec 2012)
2. Lydia, M., Kumar, S.S., Selvakumar, A.I., Kumar, G.E.P.: A comprehensive review on wind turbine power curve modeling techniques. Renewable and Sustainable Energy Reviews **30**, 452 – 460 (2014)
3. Russell, S.J., Norvig, P.: Artificial Intelligence - A Modern Approach. Pearson Education (2010)
4. Tuballa, M.L., Abundo, M.L.: A review of the development of smart grid technologies. Renewable and Sustainable Energy Reviews **59**, 710 – 725 (2016)

Smart Lifecycle Management for Devices in the Internet of Things – A Research Approach

Dominik Grüdl

Coburg University of Applied Sciences and Arts, Friedrich-Streib-Str. 2, 96450 Coburg, Germany
dominik.gruedl@hs-coburg.de

Abstract. The heterogeneity of IoT devices, networks and platforms is identified as the main problem for feasible device lifecycle management. As a possible starting point for my dissertation this summary proposes a research approach for the smart management of device lifecycles for the Internet of Things.

Keywords: Internet of Things, Lifecycle Management, Artificial Intelligence.

1 Relevance and problem definition

The Internet of Things (IoT) consists not only of physical devices that gather information about their environment, but also of their digital twins. An IoT platform enables its users and developers to manage those virtual device copies. Therefore all states, state changes and gathered information of the IoT devices need to be sent to the IoT platform via global information and communication networks. Vice versa, modifications on the digital twins need to be communicated to the physical devices.

Heterogeneity is a main attribute of the IoT. Setup and type of the IoT device depend heavily on its respective purpose and the therefore used technologies [1]. The scenario also affects the decision of which information and communication network is used. Accordingly, data from different device types with individual lifecycles and states may have to be mapped via different networks in the IoT platform.

In addition to various setups and types of devices and networks, there is a vast number of versatile IoT platforms that are designed to perfectly support any scenario. The digital twins and the current state of the physical devices are a necessity for each use case. Because of the arbitrarily high complexity of each IoT scenario it's not feasible to manually map every possible device lifecycle and state to the IoT platform.

2 Research goals

Lifecycles and states are widely used and constantly refined in computer science, e. g. as state machines or as session states in distributed systems. The lifecycles and states of IoT devices vary for each use case and depend on the used hardware as well as their specific purpose. Therefore it is necessary to define a generic model that is able to represent the lifecycles, states and correlations of any IoT device to any IoT platform.

An IoT platform should be able to provide the current lifecycle and state of any device to any connected application. To evaluate the previously defined lifecycle model the model has to be implemented and tested in an IoT platform. Device management can be used as a starting point for this approach, as it is well-defined part of the architecture for IoT platforms.

Based on the generic lifecycle model and artificial intelligence (AI) an IoT platform could manage the current lifecycle and state of any device in a smart way, even without permanent connections to the devices. Lifecycles and states could be aggregated and analyzed to extract patterns about the devices and the use case.

3 Methodology and related work

In order to work on the research questions a proper understanding of the basics of IoT and AI are required. Based on the fundamentals of IoT and IoT platforms a generic lifecycle model can be defined [2]. The next step consists of implementing, testing and evaluating the model in the context of device management in IoT platforms [3]. Finally, suited AI methods for improving the implemented lifecycle management will be examined [4]. Fig. 1 depicts the approach for the upcoming investigations.

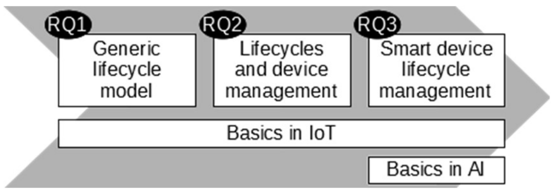
RQ1
Generic lifecycle model
RQ2
Lifecycles and device management
RQ3
Smart device lifecycle management
Basics in IoT
Basics in AI

Fig. 1. Approach for the dissertation (RQ = Research Question)

4 References

1. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of things: A survey on enabling technologies, protocols, and applications. In: IEEE communications surveys & tutorials, 17(4), pp. 2347–2376 (2015).
2. Soos, G., Kozma, D., Janky, F. N., Varga, P.: IoT Device Lifecycle – a Generic Model and a Use Case for Cellular Mobile Networks. In: 2018 IEEE 6th International Conference on Future Internet of Things and Cloud, pp. 176–183, Barcelona (2018).
3. Datta, S. K., Bonnet, C.: A Lightweight Framework for Efficient M2M Device Management in oneM2M Architecture. In: 2015 International Conference on Recent Advances in Internet of Things (RioT), pp. 1–6, Singapore (2015).
4. Mahdavinejad, M. S., Rezvan, M., Barekatain, M., Adibi, P., Barnaghi, P., Sheth, A. P.: Machine learning for Internet of Things data analysis: A survey. In: Digital Communications and Networks, 4(3), pp. 161–175 (2018).

Application Modernization: Refactoring to Microservices

Jonas Fritzsch[1], Stefan Wagner[1], and Alfred Zimmermann[2]

[1] University of Stuttgart, Stuttgart, Germany
{jonas.fritzsch, stefan.wagner}@informatik.uni-stuttgart.de
[2] University of Applied Sciences Reutlingen, Reutlingen, Germany
alfred.zimmermann@reutlingen-university.de

Abstract. While the recently emerged Microservices architectural style is widely discussed in literature, it is difficult to find clear guidance on the process of refactoring monolithic legacy applications. Software architects facing this challenge are in need of selecting an appropriate strategy and refactoring technique. Our research aims to provide means to facilitate the migration of monolithic applications to a Microservices architecture. In particular, we focus on the decomposition into services and result evaluation of a refactoring. By investigating existing refactoring approaches we identified a lack of practically applicable approaches. A subsequent empirical industry study yielded the decomposition as a main challenge. Only a fraction of the interviewed participants used a systematic approach. We aim to develop a novel decomposition approach that addresses the shortcomings of existing approaches while equally focusing on practical applicability.

Keywords: Microservices, Software Architecture, Modernization, Refactoring, Decomposition

1 Introduction

In the course of modernizing applications, several companies migrate their legacy systems towards a Microservices architecture. They primarily aim for better maintainability, shorter release cycles, scalability, cloud-readiness, or high availability. The importance of a well planned migration arises from its high cost, long duration, and involved organizational restructurings next to the architectural refactoring itself. A recent mapping study confirms a strong industry interest in migrating legacy systems [2]. The decomposition of a monolith into Microservices and especially determining the right granularity hereby is described in literature as being more of an art than a science. Studies also state that industrial state-of-practice has already reached some degree of maturity, while academia is still at an early stage. Moreover, Jamshidi et al. note that recently published papers have had "*little if any impact on microservice practice*" [4]. Our efforts aim to fill this gap in scientific research as expressed by the following four questions:

RQ1: What existing architectural refactoring techniques are applicable in the context of decomposing a system into Microservices?

RQ2: What are applied strategies and challenges in a migration process?

RQ3: How can a practically applicable decomposition method relying on static code and dynamic runtime analysis evolve from existing approaches?

RQ4: What metrics, tools, and processes can be used for evaluating the appropriateness of service partitioning and service granularity?

2 Contributions and Preliminary Results

To address RQ1, we investigated 10 refactoring approaches for Microservices recently proposed in academic literature [3]. The approaches were classified by the underlying decomposition technique and visually presented in the form of a decision guide. As a result, the practical applicability of the investigated approaches was limited by partly significant amounts of required input data, restriction to certain types of applications, or missing tool support.

To address RQ2, we contributed a qualitative study on intentions, strategies, and challenges in the context of a migration to Microservices [1]. We investigated the migration process of 14 systems across different domains and sizes by conducting 16 in-depth interviews with software professionals from 10 German-based companies. Due to the high complexity of their legacy systems, most companies preferred a rewrite using current technologies over splitting up existing code bases. This was often caused by the absence of a suitable decomposition approach. As such, finding the right service cut was a major challenge.

In the course of our future research on RQ3 and RQ4 we aim to develop a novel and practically applicable decomposition approach that combines results from source code analysis, development history (e.g. coupled change analysis), and meaningful workload data gathered during typical operation.

References

1. Bogner, J., Fritzsch, J., Wagner, S., Zimmermann, A.: Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality. In: 2019 IEEE International Conference on Software Architecture Companion (ICSA-C). pp. 187–195. IEEE (2019). https://doi.org/10.1109/ICSA-C.2019.00041
2. Di Francesco, P., Lago, P., Malavolta, I.: Architecting with microservices: A systematic mapping study. Journal of Systems and Software pp. 77–97 (2019). https://doi.org/10.1016/j.jss.2019.01.001
3. Fritzsch, J., Bogner, J., Zimmermann, A., Wagner, S.: From Monolith to Microservices: A Classification of Refactoring Approaches. In: Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment. pp. 128–141. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-06019-0_10
4. Jamshidi, P., Pahl, C., Mendonca, N.C., Lewis, J., Tilkov, S.: Microservices: The journey so far and challenges ahead. IEEE Software **35**(3), 24–35 (2018). https://doi.org/10.1109/MS.2018.2141039

A Formal Security Analysis of Hyperledger Fabric

Mike Simon and Ralf Küsters

Institute of Information Security
University of Stuttgart
Stuttgart, Germany
{mike.simon,ralf.kuesters}@sec.uni-stuttgart.de

1 Motivation

As stated recently [3], many blockchain protocols claimed security properties without formally proving or analyzing their claims. Even in some cases where researchers carried out formal security proofs, blockchain protocols do not always achieve expected properties, for example, the case of Monero [1]: the initial privacy definition was not strong enough to handle several attacks that allow revealing parties in transactions [12, 18]. This example shows that carrying out security proofs for blockchain protocols is a challenging task.

Hyperledger Fabric [7] or short Fabric is a permissioned blockchain initially developed by IBM then donated to the Hyperledger Project [9] and now open-source. Fabric is maybe one of the most important business blockchains. Several production systems use Fabric as backbone [5, 8, 10].

Fabric introduced many interesting concepts to permissioned blockchain protocols such as the *execute-validate-order* paradigm that enables higher throughput at the cost of a degree of decentralization and a modular approach for consensus algorithms that allows using several existing and well-known consensus algorithms from the distributed systems sector such as RAFT [11, 15], PBFT [4, 17], and an Apache Kafka-based consensus algorithm [14]. Indeed, the Kafka-based consensus is the most mature implementation and usually recommended for production usage. Thus, we decided to focus on Fabric using a Kafka-based consensus algorithm.

Though the construction of Fabric seems to be sound, there is no full-fledged security analysis of Fabric published so far. Androulaki et al. [2] describe the properties Fabric should achieve: *agreement* or *consistency* (honest participants should share a common prefix if one compares their chains), *hash chain integrity* (a block includes the hash of its predecessor), *no skipping* (blocks are delivered in consecutive order), *no creation* (all transactions are initiated by registered clients), and *validity* or *liveness* (transactions will be eventually part of the chain). Further, Androulaki et al. argue why Fabric should achieve these properties.

2 Our Fabric Security Analysis

We modeled Fabric in the *IITM model* [13] (IITM stands for inexhaustible interactive Turing machines), a UC model that can be used to prove common security properties of blockchains such as consistency as introduced by Pass et al. [16]. We also analyzed the desired Fabric properties according to Androulaki et al. [2].

The model includes all relevant Fabric roles, a detailed model of Kafka where we abstracted the underlying Apache ZooKeeper [6] away, and an attacker model that allows adaptive corruption. Our model includes *chaincodes* (smart contracts) and *endorsing policies*, the rule set that defines who is allowed to call which chaincode and how many "votes" a transaction needs to be accepted in the blockchain.

Unfortunately, because Kafka only achieves *crash fault-tolerance* and not *Byzantine fault-tolerance*, our analysis did not output satisfying results: we need strong and unrealistic assumptions to carry out security proofs. Basically, we need to assume that the whole Kafka cluster works honestly and a majority of the orderers behaves honestly as well.

We are working on a solution to mitigate the mentioned weakness and to improve our security proofs.

The poster presents the current state of the security analysis and depicts the relevant components for the security analysis.

Acknowledgement. This research was partially funded by the Ministry of Science of Baden-Württemberg, Germany, for the Doctoral Program "Services Computing".

Keywords: Hyperledger Fabric, Formal Security Analysis, Blockchain, Smart Contracts

References

1. Alonso, K.M., Herrera-Joancomartí, J.: Monero - Privacy in the Blockchain. IACR Cryptology ePrint Archive 2018, 535 (2018)
2. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A.D., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolic, M., Cocco, S.W., Yellick, J.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018. pp. 30:1–30:15. ACM (2018)
3. Cachin, C., Vukolic, M.: Blockchain Consensus Protocols in the Wild (Keynote Talk). In: 31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria. LIPIcs, vol. 91, pp. 1:1–1:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)

4. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. 20(4), 398–461 (2002)
5. Group, E.: Erste in Europa zur Gänze auf Blockchain basierende Kapitalmarktemission erfolgreich auf den Markt gebracht. https://www.erstegroup.com/de/news-media/presseaussendungen/2018/10/23/papierlose-ssd-blockchain (2019), (Accessed on 02/13/2019)
6. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.: ZooKeeper: Wait-free Coordination for Internet-scale Systems. In: 2010 USENIX Annual Technical Conference, Boston, MA, USA, June 23-25, 2010. USENIX Association (2010)
7. Hyperledger: Hyperledger fabric. https://www.hyperledger.org/projects/fabric (2018), (Accessed on 03/02/2019)
8. Hyperledger: Hyperledger: Finance. https://www.hyperledger.org/resources/industries/finance (2018), (Accessed on 04/29/2019)
9. Hyperledger: Hyperledger: Open source blockchain technologies. https://www.hyperledger.org/ (2018), (Accessed on 04/29/2019)
10. Hyperledger: Six Hyperledger Blockchain Projects Now in Production. https://www.hyperledger.org/blog/2018/11/30/six-hyperledger-blockchain-projects-now-in-production (2018), (Accessed on 02/13/2019)
11. Hyperledger: fabricdocs: Introduction. https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html (2019), (Accessed on 04/01/2019)
12. Kumar, A., Fischer, C., Tople, S., Saxena, P.: A Traceability Analysis of Monero's Blockchain. In: Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10493, pp. 153–173. Springer (2017)
13. Küsters, R.: Simulation-Based Security with Inexhaustible Interactive Turing Machines. In: Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006). pp. 309–320. IEEE Computer Society (2006), see http://eprint.iacr.org/2013/025/ for a full and revised version.
14. Narkhede, N., Shapira, G., Palino, T.: Kafka: The definitive guide (2016)
15. Ongaro, D., Ousterhout, J.K.: In Search of an Understandable Consensus Algorithm. In: 2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014. pp. 305–319. USENIX Association (2014)
16. Pass, R., Seeman, L., Shelat, A.: Analysis of the Blockchain Protocol in Asynchronous Networks. In: Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10211, pp. 643–673 (2017)
17. Sukhwani, H., Martínez, J.M., Chang, X., Trivedi, K.S., Rindos, A.: Performance Modeling of PBFT Consensus Process for Permissioned Blockchain Network (Hyperledger Fabric). In: 36th IEEE Symposium on Reliable Distributed Systems, SRDS 2017, Hong Kong, Hong Kong, September 26-29, 2017. pp. 253–255. IEEE Computer Society (2017)
18. Yu, J., Au, M.H.A., Veríssimo, P.J.E.: Re-thinking untraceability in the CryptoNote-style blockchain. IACR Cryptology ePrint Archive 2019, 186 (2019)