Self-aware systems for the Internet-of-Things

Mischa Möstl, Johannes Schlatow, Rolf Ernst*
Institute of Computer and Network Engineering
TU Braunschweig, Germany
{moestl,schlatow,ernst}@ida.ing.tu-bs.de

Henry Hoffmann
Systems Group, University of Chicago
hankhoffmann@cs.uchicago.edu

Arif Merchant, Alexander Shraer
Google Research
Google, Inc.
{aamerchant, shralex}@google.com

ABSTRACT

The IoT will host a large number of co-existing cyber-physical applications. Continuous change, application interference, environment dynamics and uncertainty lead to complex effects which must be controlled to give performance and application guarantees. Application and platform self-configuration and self-awareness are one paradigm to approach this challenge. They can leverage context knowledge to control platform and application functions and their interaction. They could play a dominant role in large scale cyber-physical systems and systems-of-systems, simply because no person can oversee the whole system functionality and dynamics.

IoT adds a new dimension because Internet based services will increasingly be used in such system functions. Autonomous vehicles accessing cloud services for efficiency and comfort as well as to reach the required level of safety and security are an example. Such vehicle platforms will communicate with a service infrastructure that must be reliable and highly responsive. Automated continuous self-configuration of data storage might be a good basis for such services up to the point where the different self-x strategies might affect each other, in a positive or negative form.

This paper contains three contributions from different domains representing the current status of self-aware systems as they will meet in the Internet-of-Things and closes with a short discussion of upcoming challenges.

1. INTRODUCTION

Self-awareness as a means to cope with complexity has already been proposed for Autonomic Computing, more than a decade ago. In their seminal 2001 paper [17], Kephart and Chess from IBM envision "computing systems that can manage themselves given high-level objectives from administrators". Required capabilities already include self-configuration, self-optimization, self-healing, and self-protection against defined security attacks.

Autonomic Computing addresses large scale computer systems, as used in enterprise computing which are continuously controlled and maintained by humans, the administrators. Autonomic Computing supports administration by automated diagnosis and offloads from detailed knowledge of functions and dependencies. Since then, many new contributions extended the role of maintaining these systems in an application-centric way. Applications interact with networked computing systems using control-theoretic mechanisms to optimize Quality of Service (QoS) for a variety of objectives. Self-awareness was used in the formation of virtual platforms supporting systems integration, increased system dynamics, and openness [21] preparing the basis for today's big data applications.

A major step in the development of self-aware solutions was the extension to networked embedded systems which are not supervised by humans. New paradigms, such as organic computing where proposed emphasizing self-control based on objectives derived from complex models of self-awareness or leading to emergent behavior [18, 27]. Physical system properties, such as media quality, automatic control quality, or energy consumption where added to account for embedded systems functions and resource constraints. Integration of a large variety of largely independent functions, as found in vehicles, leads to application and platform dependencies requiring self-configuration, self-optimization, and self-protection previously only known from large-scale computing systems. Safety-critical and high-availability applications, for example in autonomous driving or space robotics, have reached a level of complexity, interaction, and dynamics where classical lab-based design is not sufficient any more and should be supported or even replaced by in-field self-aware functions. Here, QoS contracting mechanisms are added to self-awareness and combined with monitoring to ensure adherence to required guarantees.

The Internet-of-Things will bring the two worlds of large-scale computing systems for big data applications and networked embedded systems together entailing interaction of the two lines of self-awareness that have individually developed over the past decade. This paper contains three contributions from different domains representing the current status of self-aware systems as they will meet in the Internet-of-Things: "Coordinating Self-aware Applications with Self-aware Resource Manager" (Sec. 2) by Henry Hoffmann University of Chicago, "Controlling Concurrent Change - A self-aware infrastructure for continuous change and evolu-

*This work was funded in part by DFG FOR 1800.

CODES/ISSS '16, October 01-07 2016, Pittsburgh, PA, USA

ISBN 978-1-4503-4483-8/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2968456.2974043

tion in automotive systems" (Sec. 3) by Rolf Ernst, TU Braunschweig, Germany, and "A big data approach to optimizing storage" (Sec. 4) by Arif Merchant and Alexander Shraer, Google Research.

2. COORDINATING SELF-AWARE APPLICATIONS WITH SELF-AWARE RESOURCE MANAGERS

Power and energy constraints are dominating the design of modern computer applications and systems. *Self-aware* computing has arisen as a broad set of techniques for adapting application and system behavior to support developers who must deal with multiple, often conflicting constraints; *e.g.*, achieving high performance and low power consumption. Two complementary self-aware approaches have developed for dealing with power and performance constraints. At the application-level frameworks have been developed for creating *accuracy-aware* applications, which can sacrifice the quality of their result for increased performance (or reduced energy usage) [24, 3, 31, 2, 14, 32]. At the system-level frameworks have been proposed to create *power-aware* systems, which respond to reduced computational demand by decreasing power or energy consumption [30, 23, 13, 33, 16, 19].

These self-aware frameworks benefit users by automatically adjusting to meet constraints. For example, accuracy-aware applications may reduce accuracy to maintain performance despite reduced resource availability. Similarly, power-aware systems respond to reduced workload by reducing resource usage. Given the potential benefits of these approaches, it is increasingly likely that accuracy-aware applications will be deployed on power-aware systems. Therefore, it is important to study their potential interaction. Recent work in this area demonstrates the importance of *actively corrdinating* decisions at the application level (such as switching to a higher-performance, lower-accuracy algorithm) with decisions at the system level (such as switching to a lower-performance, lower-power configuration) [11, 12, 10].

There are several benefits of active coordination of accuracy-aware applications with power-aware systems. Specifically, active coordination:

- Avoids oscillations and destructive interference that can arise when application and system act independently.
- Allows control of multiple dimensions simultaneously.
- Produces better outcomes for the same constraints.
- Provides a richer tradeoff space for reacting to user goals and environmental changes.

2.1 Experimental Setup

We demonstrate the benefits of active coordination by running an accuracy-aware video encoder (made with the PowerDial framework [14]) on a SandyBridge Linux x86 platform with a power-aware runtime [13]. The video encoder can reduce accuracy (adding noise to the encoded video) in exchange for increased performance. The system can increase resource usage to increase performance at a cost of increased power consumption. The following examples show the benefits that can be achieved through active coordination of the accuracy-aware application with the power-aware system.

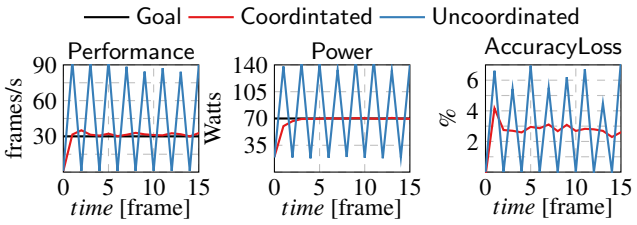

Figure 1: Running an adaptive video encoder on an adaptive system with and without coordination. The goal is to maintain 30 frames per second performance. Without coordination, the performance goal is not met while power and accuracy fluctuate wildly. In contrast, the coordinated approach meets the performance goal while conserving power and accuracy.

2.2 Avoiding Oscillations & Controlling Multiple Dimensions

Given a performance goal (*e.g.*, real-time or quality-of-service constraint), both the application and system are provably convergent (*i.e.*, they will achieve the goal) when run in isolation. When deployed concurrently, however, they may interfere with each other, resulting in constraint violations and unpredictable behavior. These problems arise because both application and system assume that changes in the other (*i.e.*, resources or workload) are rare, and will be long-lasting when they do occur. When application and system continuously react to each other, they miss performance goals, waste power, and lose accuracy.

Figure 1 shows this bad behavior and how it can be overcome through active coordination. The figures shows that the accuracy-aware application and power-aware system produce oscillating behavior when they act without coordination. However, coordination (in this case, achieved through a specially designed adaptive feedback control system [11]) drives performance and power to the desired targets (30 frames/s and 70 Watts, respectively) while avoiding oscillations.

2.3 Better Outcomes for the Same Constraints

In addition to preventing bad behavior, coordination can achieve better outcomes for the same constraints as shown in Figure 2 [12]. We first consider the energy required to perform video encoding with an accuracy constraint. As shown in the figure (left side) coordination produces lower energy for the same accuracy constraint. The results on the right side of the figure show that for an energy efficiency goal, coordination produces a smaller accuracy loss.

2.4 Adapting to Application Phases

We use our video encoder to compress a video with three distinct scenes. The top row of charts in Figure 3 illustrates the behavior of these scenes using the application's and system's default settings (*i.e.*, encoding with the highest accuracy and all system resources). The three different scenes (each 500 frames) are demarcated by the vertical dashed

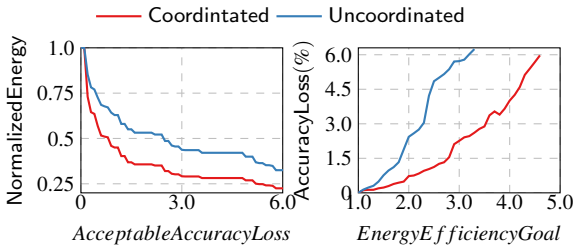

Figure 2: Coordinating application and system provides lower energy for the same acceptable accuracy (left) and lower accuracy loss for the same energy target (right).

lines. Each scene has distinct characteristics. The first is the slowest, has lowest average power consumption and produces the largest bitrate. The second scene is faster and naturally has lower bitrate. The third scene is the fastest, but occupies a middle ground in bitrate. To show the benefits of coordination, we set a soft real-time performance goal (of 30 frames per second) and deploy the encoder with a coordinated runtime, a system-level runtime, and an application-level runtime.

The bottom row of charts in Figure 3 shows how coordinated, application-only, and system-only approaches react to the changing scenes. Clearly all three approaches meet the performance goal (30 frames/s) (with some short, deviations appearing at the scene changes); however, each approach achieves the goal differently, resulting in different power consumptions and accuracies. Clearly, the coordinated approach produces the lowest power consumption of the three, achieving almost half the power consumption of the application-only approach. Furthermore, coordination also produces slightly better accuracy than an application only approach.

2.5 Conclusion & Future Work

These results indicate great potential for coordinating the adaptation of accuracy-aware applications and power-aware systems. Therefore, we propose these results should be extended by 1) exploring additional applications and systems, 2) developing formal models to describe and control the interaction between these two components, and 3) integrating frameworks which perform static analysis of these interactions with other approaches that perform dynamic runtime coordination.

3. CONTROLLING CONCURRENT CHANGE – A SELF-AWARE INFRASTRUCTURE FOR CONTINUOUS CHANGE AND EVOLUTION IN AUTOMOTIVE SYSTEMS

Safety-critical embedded-system design, as for instance practiced in the automotive domain, follows the V-model development process. This assumes that at the very beginning of the development process all requirements are specified on system level and subsequently the system is refined to sub-systems, modules, and individual functions. Implementation of software and hardware happens at the bottom tip of the V followed by corresponding integration and tests on the ascending branch of the V. However, this design philosophy and how it is applied, has a distinct shortcoming: Since design is system centric, incremental change on application and function level requires to traverse the complete (system centric) V-process, including system-level integration tests. A research group at TU Braunschweig involving 9 faculties from computer science, information technology, automotive and space science, collaborates on an approach to handle concurrent change of a system in the field thereby providing the guarantees needed for safety-critical system design. A main constraint is the compatibility to existing design processes, such that practical application without disruptive change is eventually possible.

This approach, "Controlling Concurrent change" (CCC) intends to automate the integration process on the right branch of the V design process while leaving the design of individual functions in the lab. The link between the two branches is built upon contracting mechanisms that require system self-awareness for both the contracting process and the model updates. The CCC mechanisms provide self-control for the system as its mechanisms govern what applications and functions can be integrated or updated as well as what platform changes are permissible.

In the following, Section 3.1 elaborates how the CCC architecture is structured and sketches the mechanisms used to enable automated and unattended integration capabilities. Section 3.2 then highlights how self-awareness is enabled by the CCC approach.

3.1 Architectural Approach

The CCC architecture is composed of two segregated domains, the model domain and the execution domain, forming a platform that is intended to provide safe change capabilities based on mechanisms implemented in both domains. As in any conventional system architecture, the execution domain provides the run-time environment (RTE) including the operating system (OS) required for hosting multiple application components. Furthermore, the execution domain itself is deployed on several (networked) platform components. Since all application and platform components as well as the underlying network(s) are subject to change, the model domain is introduced which employs formal methods to control these changes. In order to close the gap between assumptions made in the model domain and the actual behavior of the execution domain, the system is augmented with application and platform monitoring capabilities that not only act as protection layers but also extract metrics taken into account for future changes.

3.1.1 Model Domain

The model domain, implemented by the *Multi-Change Controller (MCC)*, is one crucial part of the CCC architecture as it plans and controls the integration process, i.e. the right branch of the V-model, which contains all necessary acceptance and conformance tests. In order to conduct these tests the MCC must have a) a sufficient mechanism to test and evaluate a certain property and b) sufficient input data available to perform an analysis with the provided mechanisms.

In our approach the set of available components and their requirements are specified by means of a so-called contracting language [15] that collects all the assumptions a component makes towards platform and run-time environment. This implicitly also includes the requirements to other applications, since the CCC mechanisms must guarantee a sat-

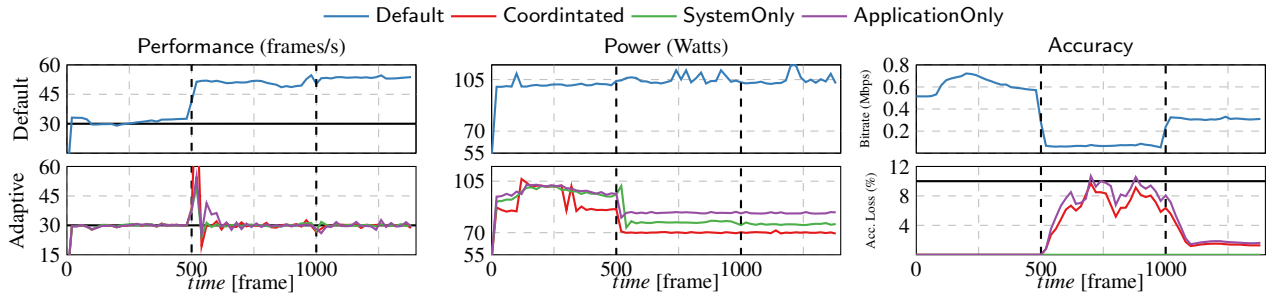

Figure 3: Comparison of non-adaptive behavior and several adaptive schemes adjusting to phases in x264.

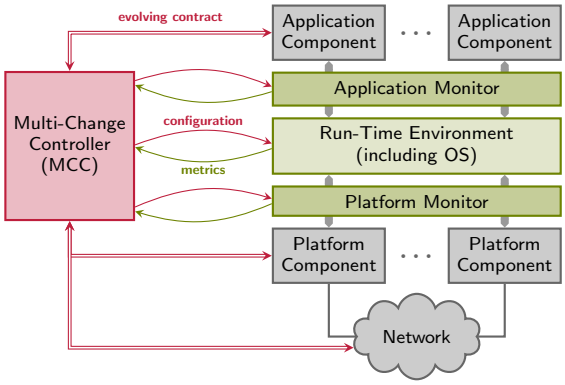

Figure 4: CCC architecture comprising a model domain (red), an execution domain (green) as well as changing application/platform components (gray).

isfactory platform sharing for the all applications. In mixed-critical systems this can mean that for certain application a guarantee in the form of a best-effort fulfillment of requirements can be satisfactory while not necessarily being safe, while other applications demand hard bounds and freedom from interference as required by safety standards. Most notably in this context are applications with different safety-level requirements.

Furthermore contracts must also specify the guarantees a component provides, given that a specified set of its requirements is fulfilled. This fact allows to compose a system from components by combining components with compatible contracts such that a desired functionality is achieved. Note that contracting is only the vehicle to formally capture requirements and must therefore be complemented by formal analysis methods that analyze whether all those requirements hold for the composed system.

The MCC keeps a repository of all available contracts as well as the current system configuration, i.e. which components are installed currently and with which parameters. Any prospective update to the system configuration invokes the MCC with a change request that generates possible configuration candidates based on the contract information. These configuration candidates further undergo several admission tests to guarantee that all requirements (functional and non-functional) are satisfied. For instance if a component requires an upper-bounded response time, this property must be checked in the complete system composition by triggering the corresponding analysis engine. In our MCC implementation, we resort to compositional performance analysis to verify contracted timing requirements of components [25].

In embedded systems, changes often require reconfiguration and optimization of the Run-Time Environment (RTE). Self-configuration mechanisms are employed that optimize software module selection and configuration [22, 26]. Furthermore, as this relies on the adherence of the components implementation to their contracts, the monitoring mechanisms can be configured to protect the system against non-conformant behavior where necessary. Suitable configuration data for the monitoring mechanisms can either be obtained from the contracts directly, e.g. for execution times of a component, or are derived from the results of analysis engines, e.g. activation patterns in case of (timing) performance analysis. This data can be used by the MCC to then configure the execution domains monitoring facilities.

3.1.2 Execution domain

The execution domain is based on the Genode OS Framework that provides a customizable run-time environment on top of several microkernels. The microkernel-based design of the execution domain provides the foundation for a safe, reliable, and secure platform sharing. Its strong isolation of application components reduces mutual interference (e.g. by fault propagation) and allows for fine-grained access control while still providing a high degree of flexibility required for in-field changes. This is a clear contrast to other monolithic (i.e. a wide range of functionality is implemented in the kernel, e.g. memory management) kernel based designs such as Linux or small statically configured systems such as AUTOSAR.

As mentioned above, certain parameters of the system can be monitored by application and platform monitors in order to assert that the execution domain behaves as expected by the model domain. Monitors for instance supervise the execution time of a component or police the access to network resources [9], this enables that either feedback can be given to adapt models of the MCC to the actual execution behavior or to perform enforcement of parameters determined by the MCC in the execution domain.

3.2 Self-awareness in CCC

The architectural approach presented above lays the foundation for adding self-awareness to complex systems by modeling and observing all relevant aspects of the system's behavior. In the scope of CCC, we particularly focus on self-protection and self-organization capabilities. By self-protecting, we refer to the ability of the system for protecting itself against changes that might impair its proper operation as well as protecting itself against non-conformant behavior

w.r.t. the models. Note that this is addressed by the MCC and the application/platform monitors respectively as discussed earlier.

However, the self-protection capability already represents a challenging task as it needs to consider a multitude of aspects. We therefore developed a modular architecture of the MCC that resembles the multi-viewpoint approach discussed in [15]. The approach uses formalized contracts to specify requirements for individual components. Depending on the nature of the requirements for individual components or sets of components they are accounted to different viewpoints. While certain components only have functional requirements, i.e. that other components must be present as well to provide a specified service, others might have strict deadline, i.e. timing requirements for an end-to-end chain of components. Again in certain cases such requirements might be relevant for the functions' safety, e.g. in control applications. Consequently these aspects – in the above example functionality, timing, and safety – constitute the viewpoints. Due to the heterogeneity of the viewpoints we adopt a federated approach of fulfilling the individual requirements rather than a global holistic solution where all requirements are forced into one holistic problem.

Thus, each viewpoint is represented by an analysis engine that performs an admission test on any prospective system configuration. Only if all admission tests have been passed, the new configuration can be deployed. Consequently, this builds a basic safeguard mechanism that allows performing in-field updates in a safe and controlled manner, which we consider as a foundation for realizing self-organization capabilities.

More precisely, the key idea is to incrementally change the system's configuration upon internal or external change requests, e.g. to realize an automatic deployment of security updates that allows the system to evolve to a more secure state. For this purpose, the system must be aware of all its components and their requirements to anticipate how a change might affect their operation. First of all, requested changes are performed such that the system automatically detects the dependencies of already instantiated components as well as the dependencies that would be introduced by the requested change. This means that upon change requests, it is evaluated whether any guarantees already given in the currently active configuration would be impaired and on which formal grounds the guarantees of the requested change relies. From this dependency analysis step a suitable configuration for the monitoring network can be derived, allowing the system to self-*protect* it against changes that would endanger safe execution of previously MCC verified and approved components.

Note that the in-field integration of software components might require extensive design-space explorations while respecting all constraints, which is clearly a challenging task, particularly when it comes to planning, evolution and optimization. The key aspect of the CCC approach is that the MCC provides guidance and control for this process such that any design decision made during the automated integration process still undergoes the admission tests. This allows applying sophisticated algorithms whose results do not necessarily need to be trusted and thus enables offloading of those algorithms even as a cloud service.

In summary the CCC platform mechanisms implemented in the MCC are triggered by a change request, which can either originate from within the system due to observed behavior from the platform and application monitors or by an external change request, e.g. an available security update for an instantiated component. Requested components are then integrated by the MCC with full awareness of the requirements for this components as well as of those already deployed on the platform. This information is taken from the contracts supplied with the application and the current state of the system as planned and verified by the MCC, due to formal analysis by the analysis engines or through observation and evaluation of metrics obtained from the monitoring network. After planning a new configuration that includes the requested components this configuration candidate is passed to the analysis engines of the different viewpoints that perform admission tests to enable a safe operation of old and new components. Only after a configuration candidate is accepted by all view-points in the MCC it is deployed to the execution domain. By following this update path the MCC allows a requirement guided self-aware integration of new components, while protecting the system against changes that would jeopardize the execution of the already instantiated components.

4. A BIG DATA APPROACH TO OPTIMIZING STORAGE

Cloud data centers are extremely large, and comprise an enormous amount of storage. As of 2010, typical Google data centers had 10,000+ servers, each with 6 x 1 TB drives per machine [7]. Since then, data centers and their storage have likely grown bigger; for example, Facebook announced a data center capable of holding an exabyte of data per data hall in a data center, albeit intended to store infrequently accessed data [4]. Storage in these data centers is typically some form of a distributed file-system such as GFS [8], database such as Spanner [6], semi-structured storage such as Bigtable [5], and more probably, a combination of such systems. Each system can have myriad tuning knobs and options for optimization, depending upon the workloads presented and the desired levels of performance and reliability. However, the workloads presented are often unpredictable in advance, and even when their general form is understood, can be variable or change over time. This makes it quite difficult to select configuration parameters that can optimize the performance and efficiency of the systems, since the optimal parameters can change.

Workloads in such data centers can be quite variable and hard to predict in advance, and therefore it is difficult to configure storage settings optimally without continually measuring the workloads. The traditional solution to optimizing storage systems with variable workloads is to use some form of adaptive control. For example, AutoControl [20] uses real-time measurements of performance with a MIMO feedback control model to set system parameters. However, such real time measurements and related optimization is hard in a large distributed system. While modern data centers are usually instrumented to provide detailed measurements of every conceivable performance parameter, the resulting data stream can be very large. At this scale, processing the information quickly enough to respond is a challenge.

We approach data center configuration as a Big Data problem, which can be solved by the enormous computing power the large data centers provide. While the workloads

are variable, they are often periodic and roughly predictable over the long term, which allows us to process the measurement data offline and apply configuration tweaks on a daily or weekly basis. The measurement data can be enormous, but sampling and statistical aggregation are effective ways to reduce the volume of measurements we need to process, while maintaining the essential details. While it is sometimes difficult to determine exactly which metrics are most correlated with performance, Machine Learning techniques can sometimes be applied to distil useful conclusions even from a mass of data that we do not understand well. In some cases, the workload measurements can be abstracted into an analytical characterization, to which standard optimization techniques can be applied.

We briefly describe two case studies where we have applied such techniques. Given the limited space available, these descriptions are incomplete, but more details can be found in the citations.

4.1 Case study 1: *Janus*

In a storage system combining disk and flash, the amount of flash is typically quite limited, since flash is expensive compared to disk. Janus [1] is a system to automatically partition flash between workloads in a distributed storage system with flash and disk tiers, to ensure that the limited flash goes to workloads that can best use it. When created, files are first placed in a flash tier and eventually evicted to disk automatically based on a FIFO or an LRU policy. (For simplicity, we limit the discussion here to FIFO eviction only.) The policy of placing newly created files in flash is based on the observation that files tend to be accessed most relatively soon after they are created, and most files grow "cold" fairly quickly. The rate at which files grow cold varies by workload, however. To accommodate this variation, the flash tier is partitioned between file groups corresponding to workloads. The Janus optimizer periodically computes optimal flash allocations per file group based on recent measurements of the workload to the file groups; the objective is to maximize the overall fraction of reads sent to the flash tier rather than the disk tier.

Measurements of the access rates for different workloads, and of how quickly the files grow cold, is based on traces of IO activity. However, it is not feasible to consider every read in each workload, since the read rates are high and logging operations can be expensive. We instead use sparce traces from Dapper [29], an always-on system for distributed tracing and performance analysis that samples a fraction of all RPC traffic. By measuring the age of the requested data for each sampled RPC, we can populate a histogram of the rate of read operations binned by age of the data read. We construct a second histogram for each workload's files to represent the distribution of file ages, by scanning the file metadata. These two histograms can then be joined to give us a *cacheability function* for the workload, which maps the amount of flash allocated to the workload to the rate of reads absorbed by the flash storage.

Given the cacheability function for each workload and the total amount of flash available, it is straightforward to compute the optimal allocation of flash to each workload. In a nutshell, we can allocate the flash, a chunk at a time, greedily to workloads with the largest marginal benefit per byte of flash; it can be shown that this is optimal for concave cacheability curves. (A concave cacheability curve means that the marginal benefit from allocating an additional byte of flash to a workload is monotonically non-increasing; this is usually the case approximately, but can be used as an approximation in any case.)

Janus is deployed in production at Google. We evaluated the performance of the Janus FIFO optimizer by comparing it against a single FIFO eviction policy — that is, treating the combination of all the workloads as a single workload, using all the flash, and a combined FIFO eviction queue. The optimized allocation, which allocated different amounts of flash to workloads based on their past workloads, produced about 50% more hits to flash than the single combined allocation with FIFO eviction, for the combination of workloads we tested.

4.2 Case study 2: *Storage Configuration*

Distributed database systems provide scalability and high-availability by partitioning and replicating data across servers, data centers, and continents. They provide services to thousands of distinct applications, with different workloads which change over time. For example, the data of each application may be accessed from a different set of geographical locations, and the volume of requests and their composition (e.g., read-heavy vs. write-heavy) may be different.

In order to effectively utilize resources and guarantee low latency, each database in the system should be configured to match its workload. For example, there should be enough replicas to withstand the query load and they should be placed close to the clients. At the same time, replicas should be chosen in a way that allows for efficient updates, which usually involve writing data to multiple replicas.

While it is clear that a static configuration wastes resources and often under-performs, configuring each of the (many thousand) databases in a manner that matches its dynamically changing workload cannot be done manually. Manual configuration is error-prone, cannot be done frequently, and does not scale as more and more applications are using the system, especially if the storage system is available as a cloud service.

We developed an automatic optimization system [28], used with one of Google's distributed databases. The system optimizes two general aspects of database configuration: it determines replica locations and the role of different replicas in the system. While all replicas can serve client queries, only a subset participate in committing updates while others learn of state-changes after they have been committed. Furthermore, one of the replicas, called *leader*, coordinates the update protocol and participates in consistent read requests, which have to return the latest committed data.

The data available to us for the purpose of performing the optimization is, for each database, its current configuration, information about its workload, and performance metrics. The workload is continuously collected by a global monitoring infrastructure and includes the rate of every type of database operation broken down by geographical locations of the issuing clients. Note that even if the workload changes slowly, allowing to forecast future performance, performance metrics are only available for the *current* configuration. Hence, the key challenge is to predict the performance of alternative (hypothetical) configurations. One possible approach could be to repeatedly reconfigure the database and perform measurements until the best configuration is found. This is prohibitively expensive since the

number of alternatives is large, each change to the system incurs a potentially significant cost (such as shifting clients' traffic or copying large amounts of data to create replicas in new locations) and the optimal configuration changes over time with the workload. Instead, we chose an analytical approach, which we describe next.

Though the performance of alternative configurations isn't known, we can predict it using two pieces of additional information: (a) the message flow of each type of database operation given any alternative configuration (e.g., an update operation involves a message from the client to the leader, then the leader sends a message to the voting replicas and waits for a majority to respond, and finally replies to the client), and (b) network latencies between Google's data centers. Combining these two can give a good estimate of operation latencies with any alternative configuration.

Finally, we must efficiently explore the space of alternative configurations. We show different techniques, appropriate for different type of configuration parameters. If replica locations and the set of voters are fixed and we're only trying to decide on the replica that should coordinate updates (the leader), there is a small number of alternatives, which can be explored exhaustively. If, on the other hand, we need to decide which replicas should vote on commits, in addition to choosing the leader among them, exhaustive search is exponential. We observe, however, that instead of going over all subsets of voters and choosing the best leader among them, we can equivalently go over all possible choices of leaders, and for each one find the optimal subset of replicas that minimize commit (i.e, majority response) time. This yields an efficient polynomial algorithm which produces the optimal solution. In the paper [28], we show that moving the leader can reduce operation latency by more than 50% for 17% of the databases and by more than 90% for some databases. Optimizing the set of voters can further significantly reduce latency for some databases, in most cases due to the fact that this allows more freedom in choosing the leader.

If replica locations must also be determined, the problem becomes a variant of the Facility Location problem, known to be NP-Complete. We implemented a heuristic solution based on a variant of the K-Means algorithm, where the main intuition is that we want to cluster active clients, and place each replica in a centroid of client clusters. We developed two algorithms, each exploring different alternative configurations and appropriate for different type of workloads, and show that by running both algorithms and considering the configurations they find, we can efficiently identify a solution that is within 5% of the optimum achieved by exhaustive search.

4.3 Conclusions

Both of the problems we describe above involve measuring a large number of metrics across one or more data centers and processing them in an automated, continuous manner. In both cases, we applied distributed data collection, sampling or aggregation to reduce the data size, analytical methods to produce a compact workload characterization, algorithms for efficient state-space exploration, and finally performance prediction and optimization, which required insights into the distributed system being optimized.

5. SUMMARY & CONCLUSION

The three contributions highlight different but complementary aspects of self-awareness for the IoT.

The last contribution highlights the necessity of automatic self-aware reconfiguration to enable complex high-volume IoT architectures. The presented optimization mechanisms automate the configuration of a platform service, i.e. storage caches, oblivious to the needs of the applications by aggregating and evaluating a large number of metrics. The gathered monitoring data serves as input for the automatic self-reconfiguration and is performed on-line, in the running system.

Based on the fact that automatic self-aware reconfiguration is inevitable for future systems, the contribution presented in Section 2 shows that in on-line adapting systems the coordination of control is highly relevant. It demonstrated that concurrently deployed adaptivity mechanisms – even if provably convergent in isolation – require a coordination strategy to achieve their goals (i.e. power, performance, accuracy). Again also this IoT system performs the configuration and adaptation at run-time with the goal of best-effort service.

The IoT platform management presented in Section 3 aims at systems that additionally require hard guarantees even if adaptivity mechanisms are applied. It therefore adds planning and modeling to the system's infrastructure in order to control application and platform changes, which enables safety-critical applications as part of the IoT.

The three contributions together highlight a core challenge of self-aware systems in the IoT: In the IoT, networked embedded systems using self-awareness to provide real-time and safety guarantees increasingly interact with global data and computing resources. These global data bases are built upon large distributed storage systems that use big-data algorithms to self-configure data management with replication and caching. Such coexistence and resulting interaction of self-aware algorithms can lead to destructive interference and control oscillations, as the first contribution demonstrates. Therefore, coordination of self-aware mechanisms from previously independent fields of industry and research will be needed to avoid complex and uncontrollable behavior of large scale safety-critical systems.

6. ACKNOWLEDGMENTS

Arif Merchant and Alexander Shraer thank their collaborators on the two projects described in this paper: Christoph Albrecht, Nate Coehlo, François Labelle, C. Eric Schrock, Artyom Sharov, Xudong Shi, Murray Stokely, and Muhammad Waliji.

Rolf Ernst, Johannes Schlatow and Mischa Möstl thank the other members of the DFG Research Unit Controlling Concurrent Change funding number FOR 1800.

Henry Hoffmann's effort on this project is funded by the U.S. Government under the DARPA BRASS program, by the Dept. of Energy under DOE DE-AC02-06CH11357, by the NSF under CCF 1439156, and by a DOE Early Career Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

7. REFERENCES

[1] C. Albrecht et al. Janus: Optimal flash provisioning for cloud storage workloads. In *Proceedings of the USENIX Annual Technical Conference*, pages 91–102, 2560 Ninth Street, Suite 215, Berkeley, CA 94710, USA, 2013.

[2] J. Ansel, M. Pacula, Y. L. Wong, C. Chan, M. Olszewski, U.-M. O'Reilly, and S. Amarasinghe. Siblingrivalry: online autotuning through local competitions. In *CASES*, 2012.

[3] W. Baek and T. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *PLDI*, June 2010.

[4] K. Bandaru and K. Patiejunas. Under the hood: Facebook's cold storage system. https://code.facebook.com/posts/1433093613662262/-under-the-hood-facebook-s-cold-storage-system-/, 2015-05-04. Accessed: 2016-07-12.

[5] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.

[6] J. C. Corbett et al. Spanner: Google's globally distributed database. *ACM Trans. Comput. Syst.*, 31(3):8:1–8:22, Aug. 2013.

[7] A. Fikes. Storage architecture and challenges. http://static.googleusercontent.com/media/research.google.com/en//university/relations/facultysummit2010/storage_architecture_and_challenges.pdf, 2010-07-29. Google Faculty Summit talk. Accessed: 2016-07-12.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[9] M. Hamad, J. Schlatow, V. Prevelakis, and R. Ernst. A communication framework for distributed access control in microkernel-based systems. In *12th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT16)*, pages 11–16, Toulouse, France, July 2016.

[10] J. Heo and T. F. Abdelzaher. Adaptguard: guarding adaptive systems from instability. In *ICAC*, 2009.

[11] H. Hoffmann. Coadapt: Predictable behavior for accuracy-aware applications running on power-aware systems. In *ECRTS*, 2014.

[12] H. Hoffmann. Jouleguard: Energy guarantees for approximate applications. In *SOSP*, 2015.

[13] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal. A generalized software framework for accurate and efficient managment of performance goals. In *EMSOFT*, 2013.

[14] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Dynamic knobs for responsive power-aware computing. In *ASPLOS*, 2011.

[15] S. Holthusen, S. Quinton, I. Schaefer, J. Schlatow, and M. Wegner. Using Multi-Viewpoint Contracts for Negotiation of Embedded Software Updates. In *First International Workshop on Pre- and Post-Deployment Verification Techniques (PrePost)*, Reykjavik, Iceland, June 2016.

[16] C. Imes, D. H. K. Kim, M. Maggio, and H. Hoffmann. Poet: A portable approach to minimizing energy under soft real-time constraints. In *RTAS*, 2015.

[17] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan. 2003.

[18] P. R. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, J. Torresen, and X. Yao. A Survey of Self-Awareness and Its Application in Computing Systems. In *2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, pages 102–107, Oct. 2011.

[19] N. Mishra, H. Zhang, J. D. Lafferty, and H. Hoffmann. A probabilistic graphical model-based approach for minimizing energy under performance constraints. In *ASPLOS*, 2015.

[20] P. Padala et al. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 13–26, New York, NY, USA, 2009. ACM.

[21] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive Control of Virtualized Resources in Utility Computing Environments. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 289–302, New York, NY, USA, 2007. ACM.

[22] A. Reschka, M. Nolte, T. Stolte, J. Schlatow, R. Ernst, and M. Maurer. Specifying a middleware for distributed embedded vehicle control systems. In *Proceedings of the 2014 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 117–122, Hyderabad, India, December 2014.

[23] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. Energy management in mobile devices with the cinder operating system. In *EuroSys*, 2011.

[24] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: approximate data types for safe and general low-power computation. In *PLDI*, 2011.

[25] J. Schlatow and R. Ernst. Response-time analysis for task chains in communicating threads. In *22nd IEEE Real-Time Embedded Technology and Applications Symposium (RTAS 2016)*, Vienna, Austria, April 2016.

[26] J. Schlatow, M. Moestl, and R. Ernst. An Extensible Autonomous Reconfiguration Framework for Complex Component-Based Embedded Systems. In *2015 IEEE International Conference on Autonomic Computing (ICAC)*, pages 239–242, July 2015.

[27] H. Schmeck, C. Müller-Schloer, E. Çakar, M. Mnif, and U. Richter. Adaptivity and Self-organization in Organic Computing Systems. *ACM Trans. Auton. Adapt. Syst.*, 5(3):10:1–10:32, Sept. 2010.

[28] A. Sharov, A. Shraer, A. Merchant, and M. Stokely. Take me to your leader!: Online optimization of distributed storage configurations. *Proc. VLDB Endow.*, 8(12):1490–1501, Aug. 2015.

[29] B. H. Sigelman et al. Dapper, a large-scale distributed systems tracing infrastructure. Technical report, Google, Inc., 2010.

[30] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. Koala: A platform for os-level power management. In *EuroSys*, 2009.

[31] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. Eon: a language and runtime system for perpetual systems. In *SenSys*, 2007.

[32] X. Sui, A. Lenharth, D. S. Fussell, and K. Pingali. Tuning variable-fidelity approximate programs. In *ASPLOS*, 2016.

[33] M. Weiser, B. B. Welch, A. J. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *OSDI*, 1994.