Trends in On-Chip Dynamic Resource Management

Kasra Moazzemi*, Anil Kanduri†, Dávid Juhász‡¶, Antonio Miele§,
Amir M. Rahmani*‡, Pasi Liljeberg†, Axel Jantsch‡ and Nikil Dutt*
*University of California, Irvine, United States
†University of Turku, Turku, Finland
‡TU Wien, Vienna, Austria
§Politecnico di Milano, Milan, Italy
¶Imsys AB, Stockholm, Sweden
Email: moazzemi@uci.edu, spakan@utu.fi, david.juhasz@tuwien.ac.at, antonio.miele@polimi.it,
amirr1@uci.edu, pasi.liljeberg@utu.fi, axel.jantsch@tuwien.ac.at, dutt@ics.uci.edu

***Abstract*—The Complexity of emerging multi/many-core architectures and diversity of modern workloads demands coordinated dynamic resource management methods. We introduce a classification for these methods capturing the utilized resources and metrics. In this work, we use this classification to survey the key efforts in dynamic resource management.**

We first cover heuristic and optimization methods used to manage resources such as power, energy, temperature, Quality-of-Service (QoS) and reliability of the system. We then identify some of the machine learning based methods used in tuning architectural parameters in computer systems. In many cases, resource managers need to enforce design constraints during runtime with a certain level of guarantee. Hence, we also study the trend in deploying formal control theoretic approaches in order to achieve efficient and robust dynamic resource management.

I. Introduction

Dynamic resource management has been established as an effective technique to improve reliability, efficiency, and performance of computer systems [1]. Managing shared resources during runtime becomes more complex with modern multicores which support diverse workloads that exhibit varying resource demands, sometimes with conflicting limitations. This dynamic behavior of workloads which vary across concurrent applications, creates significant challenges for homogeneous architectures. The need for a holistic dynamic resource management technique becomes more vital in emerging heterogeneous multicore processors (HMPs) where heterogeneous compute units are deployed on a single chip, allowing trade-offs between objectives such as maximizing performance and minimizing power consumption [2].

In this context, computer architects use several approaches to perform dynamic resource management. Model-based and Rule-based heuristic methods use a model or an encoded algorithm to make decisions during runtime. Optimization methods minimize/maximize an objective while considering certain constraints. Machine learning methods learn the best input values for different observed conditions. Finally, control theoretic techniques, use their intrinsic feedback loop to adapt to conditions.

We present a categorization of resources used in dynamic resource management that can capture the efforts done in this area. The categories reflect the nature of resources such as tangible resources (actual hardware blocks), resource metrics (characteristics of the services realized by tangible resources) and liquid resources (characteristics of the physical state of the system). Based on this categorization, we survey selected key efforts regarding run-time management of each resource type. In addition, one promising solution for robust dynamic resource management is the use of control theoretic approaches that provide formal guarantees in response to dynamic workloads. We highlight the trend in usage of control theoretic methods for dynamic resource management and identify some of the future steps that can be done in this direction to better guarantee the control of shared resources in manycore systems.

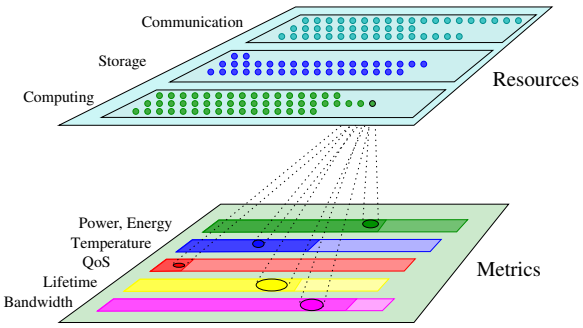

Fig. 1: Capturing the relationship between resources and metrics. The upper plane represents resources while the lower shows metrics. The allocation and usage of each resource throws shadows into the metrics plane representing how it contributes to the operation with respect to different metrics.

II. Categorization and Taxonomy

We start with an inventory of *resources* that are subject to resource management. Three sub-categories of resources can be identified according to the functionality they provide:

- **Computation resources** are processing elements which perform tasks.
- **Communication resources** are utilized by tasks to exchange information with other tasks or the environment.
- **Memory resources** are used by tasks to store and retrieve data.

Resources are physical hardware blocks that can be allocated to tasks by resource management. Allocation choices of

resource management are binary: a resource is either allocated to a task at a time or not. However, a resource may have the capacity to serve more than one task if the hardware is provided. Each resource has a maximum capacity of tasks that it can serve at a given time, which depends on its hardware structure. For instance, a CPU with two ALUs could process two instructions at the same time.

Control decisions of resource management also tune the operation of resources. The operation of the system can be characterized by various *metrics*, see Fig. 1. Note that metrics characterizing one resource are typically interdependent. For example, scaling up frequency of a computation resource results in higher power dissipation and temperate but also increases execution speed.

Resource management controls resources by allocating tasks and tuning operation parameters. All actions of resource management are based on the observed operation of the system, that is metrics. By allocating and controlling resources properly, resource management steers the system to meet objectives for optimizing some metrics. Negative consequences of over-utilizing the system needs to be avoided, too. While aiming at meeting requirements with respects to objective metrics, resource management takes constraints on other metrics into account. Heuristics that aim at containing *constraint metrics* below or above given limits are needed for realizing proper resource management. Note that metrics cannot be generally sorted into groups of objectives and constraints as their role is dependent on the considered resource management technique.

In this paper, we study some of the key efforts toward dynamic resource management in computer systems. First, **model/rule-based heuristics and optimization methods** are surveyed. Next, we take a look at **machine learning methods** focused on tuning architectural parameters in computer systems. Finally, we describe the trend where architects use **control theoretic methods** in dynamic resource management. In this context, the important properties for us are **robustness**, **formalism**, **efficiency**, **coordination**, **scalability** and **autonomy**. Robustness is defined as the ability to provide guarantees and perform robust analysis. Formalism facilitates reasoning about and synthesis of resource management strategies. Efficiency is in regard to lightweight design, yet responsiveness of controllers. Coordination is the ability to control actuations while tracking multiple objectives simultaneously. Scalability defines the proper design of control hierarchies to manage large and complex systems. Autonomy of a controller corresponds to automatic response to abrupt runtime changes in objectives.

III. Heuristics and Optimization Methods

A. Power and Energy

Nowadays, computer systems design is confronted with delivering high performance while limited with their power consumption. The diversity in the type and increasing complexity of applications demands higher computation power. To deliver this performance, designers have to consider the reasonable autonomy in battery-powered systems, operational cost of cloud servers as well as reduction in the environmental impacts of power consumption.

Dynamic Power Management (DPM) and system design with the goal of energy efficiency has been studied in details in the past decades [3]. Designers used DPM in the 90s [4] with the available run-time configurations such as scaling the supply voltage to lower the power consumption [5]. Run-time monitoring of application behavior lead to improved optimization in power consumption [6]. The trend towards multi/many core platforms required techniques that can formally guarantee power management of the system given a power budget which is addressed in [7], [8]. On the other hand, in cases where power consumption is not predictable at design time adaptive approaches such as [9] can be used. For instance [10] uses an adaptive power management technique for hard real-time systems and [11] proposes a runtime mapping for many-core systems.

B. Temperature

Delivering high performance does not only come with the cost of power consumption. Often circuits that perform in their highest computation power suffer from thermal issues such as overheating or faults due to thermal emergencies. Many dynamic management methods have this in mind during runtime and try to avoid such conditions. Thermal induced problems can appear in various forms such as hot spots [12], spatial variations [13] and temporal variations [14]. The goal of dynamic thermal management (DTM) is to address thermal hotspots or reduce spatial and temporal temperature variations. Clock frequency scaling, DVFS, Decode Throttling, Speculation control and cache toggling are some of the DTM techniques described in [15]. Temperature aware scheduling for multi-threaded processors can reduce hot spots [16]. Smart performance and power modeling can reduce the power leakage and limit temperature increase which can improve performance and power consumption [17]. Thermal management became a prominent challenge in dark silicon era [18]. Furthermore, [19] proposes a thermal-aware computation in nano scale technologies. Mechanisms in [20], [21] consider 3-D stacking architectures and the thermal limitations for such chips. Thermal aware communication systems can reduce the possibility of thermal emergencies in the system [22]. [23] proposes a thermal aware method for dynamic buffer allocation for network-on-chip based systems. In the same domain, [24] proposes a runtime workload mapping on network-on-chip based systems considering ripple effect of applications.

C. QoS

Quality-of-service (QoS) is a primary metric to qualitatively evaluate the system's efficiency in satisfying applications' requirements. Applications from different domains have different QoS metrics such as frame rate (multi-media) [25], latency-per-query (web search and financial) [26], throughput (data analytics and streaming) [27], responsiveness (user centric) [28], end-to-end latency and privacy (social media) [29], etc. Run-time QoS management becomes necessary and challenging

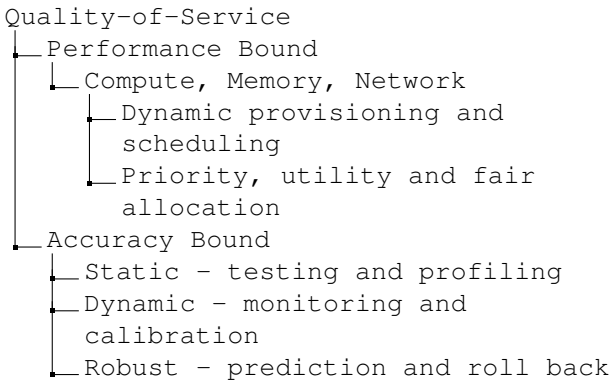

Fig. 2: Abstract classification of QoS management.

with i) variable workload characteristics ii) variable QoS requirements of applications, iii) identification and translation of QoS metrics into system level parameters for provisioning and iv) resource contention and arbitration among concurrent applications. Meeting QoS requirements of applications are largely based on:

- the nature of computation - compute, memory and I/O intensity, streaming inputs and batch processing
- the nature of end result - numerical, perceptive, soft and hard real-time, and user-interaction

We abstractly classify QoS management techniques as *performance-bound* and/or *accuracy-bound*, as shown in Figure 2. We present major underlying approaches and strategies for performance bound QoS guarantees through provisioning compute, memory and network bandwidth resources and accuracy-bound QoS through quality monitoring and control.

1) Performance Bound QoS: Performance bound QoS can be guaranteed with compute, memory, network and I/O bandwidth provisioning with dynamic priority identification.

Compute: Allocating more and/or suitable cores, CPU time slices, exploiting core-level asymmetry to fit application's QoS requirements are common approaches for QoS guarantees [30] [26] [27]. Under workload diversity, smart co-location - scheduling an optimized combination of latency and throughput sensitive applications together, exploits under-utilized resources to satisfy QoS of both types of appplications [31] [32] [30] [27] . All these techniques feature user/application defined QoS metrics such as latency and throughput bounds or dynamic identification of critical resource contending regions of code [33] and measure QoS in terms of IPC and harmonic speed up for scheduling decisions. Monitoring QoS based on IPC and satifying applications requirements through optimized time slice sharing among concurrent applications is proposed in [34]. Combining a set of cores, memory and network bandwidth into a package to provision isolated resources for applications as per their QoS requirement is proposed in [35], to provide infrastructure as a service. All the provisioning techniques prioritize applications based on QoS requirements and dynamically adapt further by monitoring resource utilization upon provisioning.

Memory and Storage: With the widening compute-memory performance gap, allocating larger cache slices and higher memory bandwidth significantly enhances performance bound QoS metrics [36] [37] [38]. Using cache partitioning to provide either larger/sufficient cache slices is a common approach to meet QoS requirements of latency critical applications [39] [40]. Identifying application/thread priority and scaling cache allocation accordingly, following utilitarian principles is another strategy to improve overall throughput metrics [41] [42] [36]. Optimizing for memory controller proximity [43] [38] and allocating higher bandwidth can enhance QoS of memory intensive applications [44] [45]. All the dynamic memory provisioning techniques however require micro-architectural/OS level extensions to identify and translate between user/application defined QoS performance metrics to system level QoS utilization metrics [46] [47] [41].

Network and I/O: Allocating higher network and I/O bandwidth to prioritized applications can guarantee latency and throughput QoS requirements. Existing techniques have used customized router architecture, virtual channels, flow control and frame scheduling to provide higher network bandwidth for dynamically identified priority applications. Classification of network into shared resource and non-shared resource clusters to allocate non-QoS and QoS tasks respectively through novel router architecture was proposed in [48]. Assigning each flow into frames and intelligent scheduling globally synchronized frames to optimize for latency is proposed in [49]. The same idea is extended by [50] with a flexible local frame scheduling and pre-emptive flit reservation for more bandwidth for high priority applications. Distinguishing between latency and throughput sensitivity of best effort (BE) and guaranteed throughput (GT) to optimize their respective flow control is proposed by [51] [52]. While BE applications are prioritized by default, priority is inverted to GT when BE applications have used enough buffer space reflecting in a certain throughput guarantee. Assigning a fixed bandwidth to each flow and monitoring its bandwidth utilization to prioritize and allocate network resources to utility frames is proposed in [53]. A similar approach with hybrid fair and elite round robin bandwidth allocation using weighed priorities is proposed in [54]. Each of these techniques dynamically determine priority of packets (originating from priority applications) and route them first, while other low priority packets wait in the queue.

2) Accuracy Bound QoS: Approximate computing leverages inherent error resilience of applications from domains such as machine learning, multi-media processing, streaming, data mining and analytics due to algorithmic nature, redundant input data, and perceptive end results for performance and energy gains [55]. However, reasoning for accuracy loss, guaranteed error bound and control on quality of result (QoR) is crucial for viability of approximation techniques. Existing approximation techniques use profiling, calibration and light-weight checks for nominal quality control [56]. We divide quality assurance techniques into three categories: viz., *static* - profiling, *dynamic* - calibration and *robust* - control and roll back.

Static Techniques: Profiling techniques validate results of an approximated code block over an exhaustive set of inputs against the accurate result to derive empirical guarantees on

error [57] [58] [59]. Static techniques are as effective as the input data coverage i.e., error bounds can be guaranteed for input sets that are pre-evaluated at profiling phase, which can in turn be used at run-time for quality control.

Dynamic Techniques: Dynamic quality control techniques use calibration - executing each candidate block of code over both accurate and approximate methods to determine nature and extent of error induced at run-time [60] [58]. These approaches then either rely on user-defined or application level accuracy requirement targets to determine whether approximate execution is within an acceptable quality range. Some techniques use the target accuracy requirement as a feedback to explore accuracy-performance Pareto space to configure the extent of approximation [58] [59]. Dynamic techniques are efficient in providing empirical and/or statistical guarantees on quality, however they require additional hardware/software overhead for continuous monitoring and execution of both accurate and approximate versions. Reducing sampling rate of monitoring might ignore errors induced during the un-sampled interval.

Robust Techniques: Robust quality control techniques monitor accuracy loss at run-time and can roll back for more accurate execution in case of errors induced beyond acceptable thresholds. Robust techniques address the limitations of static techniques which can provide guarantees only over tested inputs, and dynamic techniques which have overheads and lesser coverage within sampled invocation. Robust techniques use predictive, online learning, light-weight checks and monitoring strategies to compute quality loss and predict the extent of quality loss for subsequent inputs [56] [61]. The quality loss is compared against user defined accuracy requirements to either tone down aggressive approximation or choose a different type of approximation technique [62] [63]. In case of unacceptable results, these approaches roll back i.e., re-execute the candidate code blocks in accurate mode to cover for the accuracy loss. Robust techniques include re-configuring the extent of approximation [61] [63], re-generation of type of approximation used iteratively [64] [65] and pro-actively [66] [56] and roll back by re-executing the code block accurately [67] [68].

D. Reliability

In the last decade, reliability has become a major issue in digital circuits [69]. The aggressive scaling to nanoscale CMOS structures has caused a variety of reliability threats such as aging and wear-out acceleration due to the increased power densities and consequent thermal stress, higher susceptibility to soft errors not only in harsh environments but also at ground level, device variability leading to timing errors and other effects, etc. This issue has been even more exacerbated by the pervasiveness of computing systems in nowadays life spanning from smart environments to datafarms devoted to control and support of decision processes.

Device aging and wear-out are some of the predominant reliability issues since they cause a sensible shortening of the lifetime [70] (lifetime variation may be also quantified in 2x [71]). Unfortunately, DTM does not suffice since aging control cannot be performed only by limiting hotspots and temperature variations. Instead, as discussed in the literature [72], [73], it can only be fully achieved by monitoring the "cumulative" degradation behavior of the aging phenomena and accordingly managing resources in an aging-aware way for executing the workload. Nevertheless, this strategy is particularly effective for multi-core or many-core platforms (both homogeneous and heterogeneous ones) thanks to the availability of a large set of "programmable" processing resources, representing a sort of redundancy, that can be dynamically tuned and selected for the execution of the various applications composing the workload, that is dynamically changing as well.

The first Dynamic Reliability Management (DRM) approach [72] focusing on a single general purpose processor was proposed in 2004. After that, following also the architectural progresses in the subsequent years, different types of platforms have been considered spanning from the classical homogeneous multi-core architecture [74] [75] [76], where processing units are connected on a single bus and with a shared memory, to the NoC-based many-core architecture [77] [78] [79]. Recently, heterogeneous architectures [80] [81] [82], integrating asymmetric processors, GPUs or custom accelerators, have been also addressed in lifetime management. Depending on the specific architecture, the resource management approaches act on application mapping (as in the case of many-cores architectures [78] [83]), scheduling (as in the case of shared-memory systems [74]), and/or on power-related knobs (DVFS and per-core power gating [72] [74] [75]). Another relevant aspect is that lifetime is only one of considered parameters, thus leading in most of the approaches to a co-optimization with performance or power/energy-consumption.

Another relevant reliability issue in modern technologies is the high susceptibility to soft errors of the devices. Such transient faults use to occur with a given Soft Error Rate (SER) that is dependent also on the operating voltage/frequency levels and may be subject to variability among the various cores of the same device [84]. Therefore, runtime resource management policies [84] [85] [86] have been proposed to optimally distribute the workload and tuning architectural knobs to maximize the system reliability, measured as the probability to complete successfully current computations, together with other metrics (performance, power consumption and lifetime).

Finally, it worth mentioning a last class of runtime resource management approaches that prefer to integrate also fault handling in the controller at software level rather than using classical architectural mechanisms. To give few examples, some approaches (e.g. [87]) dynamically replicate application execution to perform fault detection or mitigation w.r.t. both transient and permanent faults, while other strategies (e.g. [88], [89]) schedule at runtime software-based self testing routines to identify permanent damages.

IV. Machine Learning Approaches

The popularity of machine learning methods has grown in the past decade. The learning nature of these methods make them a good candidate for model prediction and resource allocation in

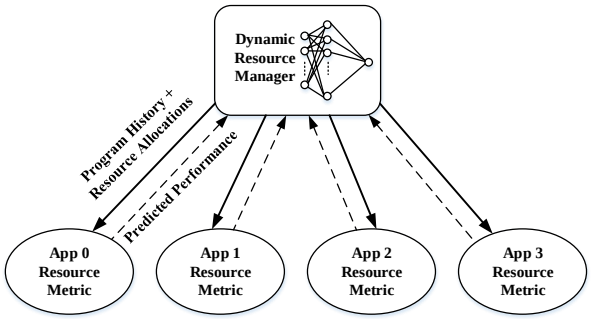

Fig. 3: High-level view of resource management technique using machine learning similar to method presented in [90].

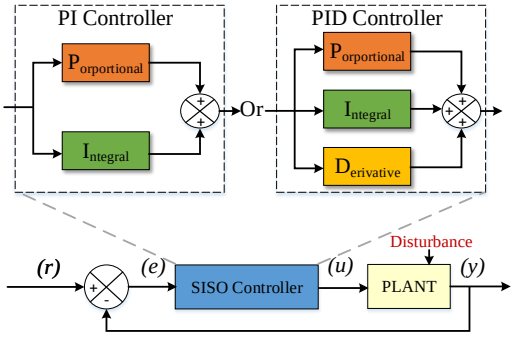

Fig. 4: *Single Input Single Output* (SISO) feedback loop.

computer system. It should be noted that these methods mostly require a learning phase with a large amount of measurement data to prepare the predicting algorithm for a specific platform. Using the advance machine learning, reinforcement learning and deep neural network methods can bring high accuracy in prediction and tuning of the architectural parameters in case that the system conditions stay the same to the conditions captured in training phase. The advantage of using these methods lie in the data driven identification of relationships that can be used for tuning system configurations at runtime.

We can categorize the machine learning methods used in resource management of computer systems based on their ability to adapt to changes over time. Data intensive and time consuming training phase of these methods often requires a detailed analysis and tuning at design time. In many cases the platform and the workloads are predetermined which gives the designers the opportunity to train the model once at design time and during execution use the **offline** model for prediction and tuning. On the other hand, **online** methods are needed to learn new changes in workload behavior or environment changes and incorporate that in the machine learning methods. These methods mostly start with a default model trained at design time and improve them at runtime to increase the prediction accuracy and management efficiency.

Figure 3 depicts a general resource management mechanism similar to the method proposed by [90]. This method leverages Artificial Neural Networks (ANN) to manage multiple shared CMP resources in a coordinated fashion to achieve a high-level objective. It's important to note that although the major part of the training phase for machine learning approaches is done at design time, the decision making and often backward learning for adaptation can be done at runtime. Authors in [91] propose using machine learning for microarchitectural adaptivity control. Approaches such as reinforcement learning have been used to design self-optimizing memory controllers [92]. Similarly, [93] proposes a dynamic resource management using deep reinforcement learning. A hybrid method is proposed in [94] to benefit from both high accuracy in modeling of machine learning methods and adaptivity of control theoretic approaches. In future, on-chip dynamic resource management can benefit from data driven identification and high accuracy of prediction in machine learning methods used along side lightweight heuristics or robust control theoretic methods [94].

V. Control Theory Techniques

Dynamic resource management for many-core systems is increasingly challenging due to the complex interactions in these systems. Integration of hundreds of cores and uncore components running various workloads with conflicting constrains increase the pressure on limited shared system resources. A promising and well-established approach is the use of control-theoretic solutions based on rigorous mathematical formalisms that can provide bounds and guarantees for system resource management [95]–[101]. In this context, we discuss some of the efforts that deploy control-theoretic centric run-time management. Starting from simple Single Input Single Output (SISO) controllers used in power management to more complex Supervisory Control Theory (SCT) methods used in dynamic resource management of complex computer systems.

A. Single Input Single Output Controllers

In the past, control theoretic methods have been proposed for resource management in the presence of a specific type of workload running on the system. A majority of these methods use Single-Input Single-Output (SISO) controllers for the ease in deployment and the guarantees they provide in tracking the target output. These SISO controllers often deploy Proportional Integral (PI), Proportional Integral Derivative (PID), or lead-lag methods. While designing SISO controllers for computer systems, often PI controllers are deployed for resource management. It is important to note that although derivative control law can be benefited in order to add predictability to the controller, stochastic variations on the system output caused by dynamic behaviour of executing applications may cause inaccuracy in the controller. Therefore, in many cases for computer systems PI controllers are preferred over PID controller [102]. PI control benefits from both integral control (zero steady-state error) and proportional control (fast transient response) [103]. Figure 4 depicts a first-order feedback SISO controller which can be deployed either as a PI or a PID controller. The error e is the input to the controller.

Note that to compute the current control input u, the controller needs to have the current value of the error e along with the past value of the error and the past value of the control

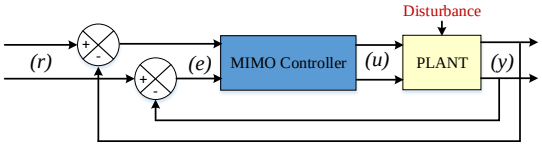

Fig. 5: Basic 2×2 Multiple-Input-Multiple-Output (MIMO).

input. It is this memory inherent in the controller that makes it dynamic.

B. Multiple Input Multiple Output Controllers for Coordination

Modern multicores support execution of diverse set of workloads with varying resource demands, which sometimes exhibit conflicting constraints. This issue exacerbates in heterogeneous multicore processors (HMPs) deploying heterogeneous compute elements on a single chip. In this context, the use of SISO controllers might not be effective as multiple system goals varying over time need to be adaptively managed and objectives holistically coordinated. Multiple-Input-Multiple-Output (MIMO) control theory is able to not only manage but also prioritize between multiple design goals. These controllers have proven to be effective for coordinating management of multiple goals in unicore processors [104] and HMPs [105]. Consider the MIMO controller in Figure 5 that controls a system with two control inputs and two interdependent measured outputs. Picking actuators and measurement metrics that result in behavior that can be estimated linearly is an important aspect of designing a stable controller [106].

C. Supervisory Control Theory

Supervisory Control Theory (SCT) [107] [108] provides formal and systematic supervision of classical MIMO/SISO controllers [109]. SCT uses modular decomposition of control problems to manage their complexity. Specifically, supervisory control has two key properties: i) rapid adaptation in response to abrupt changes in management policy and ii) low computational complexity by computing control parameters for different policies **offline**. New policies and their corresponding parameters can be added to the supervisor on demand (e.g., by upgrading the firmware or OS). Therefore, SCT is suitable for resource management problems (such as managing power, thermal, QoS, and interconnects) that can be modeled using logic and discrete system dynamics.

Figure 6 depicts a high-level view of supervisory control for many-core system resource management. Either the user or the system software may specify *Variable Goals and Policies*. The *Supervisory Controller* aims to meet system goals by managing the low-level controllers. High-level decisions are made based on the feedback given by the *High-level Plant Model*, which provides an abstraction of the entire system. Various types of *Classic Controllers*, such as PID or state-space controllers, can be used to implement each low-level controller based on the target of each subsystem. The flexibility to incorporate any pre-verified off-the-shelf controllers without the need for system-wide verification is essential for the modularity of this approach. The supervisor provides parameters such as output references or gain values to each low-level controller during runtime according to the system policy. Low-level controller subsystems update the high-level model to maintain global system state, and potentially trigger the supervisory controller to take action. The high-level model can be designed in various fashions (e.g., rule-based or estimator-based) to track the system state and provide the supervisor with guidelines. Supervisory control provides the opportunity to benefit from both classical control theoretic methods and heuristics in a robust fashion.

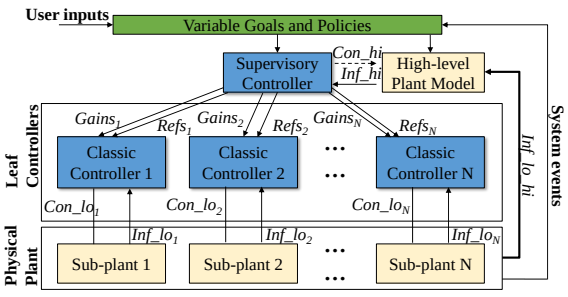

Fig. 6: High-level view of Supervisory Control Theory.

VI. Discussion

To summarize the coverage of existing on-chip resource management methods studied in this work we use Table I. Some heuristic approaches (Row A) focus on **efficiency (3)** and **coordination (4)**, but fail to provide formal guarantees and autonomy to the system. On the other hand, machine learning methods (Row B) lack robustness against corner cases. Online machine learning methods that can learn during runtime can provide better autonomy to response to abrupt runtime changes in objectives. As we have seen in Section V-A, Single-Input-Single-Output (SISO) control theoretic approaches (Row C.1) provide means to address **robustness (1)**, **formalism (2)** and **efficiency (3)**, while lacking the ability to concurrently coordinate and control multiple objectives in a non-conflicting manner. Recently-proposed Multiple-Input-Multiple-Output (MIMO) control (Row C.2) enables **coordination (4)**, addressing attributes (1) to (4). However, MIMO control lacks **autonomy (5) and scalability (6)** for complex systems. In order to address all six key challenges in dynamic resource management of complex many-core system, Supervisory Control Theory (Row C.2) has been proposed as a **scalable (5)** and **autonomous (6)**.

	Methods	1. Robustness	2. Formalism	3. Efficiency	4. Coordination	5. Autonomy	6. Scalability
A	Heuristic methods			✓	✓		
B.1	Offline Machine Learning methods		✓	✓	✓		⋆
B.2	Online Machine Learning methods		✓	✓	✓	⋆	⋆
C.1	SISO Control Theory	✓	✓	✓			
C.2	MIMO Control Theory	✓	✓	✓	✓		
C.3	Supervisory Control Theory	✓	✓	✓	✓	✓	✓

TABLE I: Major on-chip resource management approaches and the key questions they address (⋆ represnets partial coverage)

VII. Conclusion

We proposed a classification for dynamic resource management based on allocation choices and control decisions. We surveyed heuristics, machine learning and control theoretic methods used in tuning architectural parameters in computer systems. Our target resource metrics while studying these methods are power, energy, temperature, Quality-of-Service and reliability. We surveyed some of the recent efforts to increase prediction accuracy for allocation and resource management by leveraging machine learning methods. In order to provide robustness in resource management of multi/many-core systems we studied the evolution of control theoretic methods in dynamic resource management. Finally, we discussed a summary of the coverage of existing on-chip resource management methods studied in this work.

Acknowledgment

This research was partially funded by the European Unions Horizon 2020 Framework Programme for Research and Innovation under grant agreement no 674875 (oCPS Marie Curie Network).

References

[1] A. Rahmani *et al.*, *The Dark Side of Silicon*, 1st ed. Springer, Switzerland, 2016.

[2] A. M. Rahmani *et al.*, "HDGM: Hierarchical Dynamic Goal Management for Many-Core Resource Allocation," *IEEE ESL*, 2017.

[3] L. Benini *et al.*, "Policy optimization for dynamic power management," *IEEE TCAD*, 1999.

[4] E.-Y. Chung *et al.*, "Dynamic power management for nonstationary service requests," in *DATE*, 1999.

[5] L. S. Nielsen *et al.*, "Low-power operation using self-timed circuits and adaptive scaling of the supply voltage," *TVLSI*, 1994.

[6] C. Isci *et al.*, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *MICRO*, 2006.

[7] P. Bogdan *et al.*, "Dynamic power management for multidomain system-on-chip platforms: An optimal control approach," *ACM TODAES*, 2013.

[8] A. M. Rahmani *et al.*, "Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach," in *ISLPED*, 2015.

[9] M. Shafique *et al.*, "Self-adaptive hybrid dynamic power management for many-core systems," in *DATE*, 2013.

[10] K. Huang *et al.*, "Adaptive dynamic power management for hard real-time systems," in *RTSS*, 2009.

[11] A. Kanduri *et al.*, "Dark silicon aware runtime mapping for many-core systems: A patterning approach," in *ICCD*, 2015.

[12] M. Meterelliyoz *et al.*, "A leakage control system for thermal stability during burn-in test," in *ITC*, 2005.

[13] A. H. Ajami *et al.*, "Analysis of substrate thermal gradient effects on optimal buffer insertion," in *ICCAD*, 2001.

[14] A. K. Coskun *et al.*, "Temperature aware task scheduling in mpsocs," in *DATE*, 2007.

[15] D. Brooks *et al.*, "Dynamic thermal management for high-performance microprocessors," in *HPCA*, 2001.

[16] J. Donald *et al.*, "Leveraging simultaneous multithreading for adaptive thermal control," 2005.

[17] W. Liao *et al.*, "Temperature and supply voltage aware performance and power modeling at microarchitecture level," *TCAD*, 2005.

[18] R. Ayoub *et al.*, "Energy efficient proactive thermal management in memory subsystem," in *ISLPED*, 2010.

[19] H. Hajimiri *et al.*, "Proactive thermal management using memory based computing," in *NANOARCH*, 2013.

[20] S. Lee *et al.*, "Runtime thermal management for 3-d chip-multiprocessors with hybrid sram/mram l2 cache," *TVLSI*, 2015.

[21] W. H. Lo *et al.*, "Thermal-aware dynamic page allocation policy by future access patterns for hybrid memory cube (hmc)," in *DATE*, 2016.

[22] M. Wolf *et al.*, "Power and thermal modeling for communication systems," in *SiPS*, 2016.

[23] C. T. Chou *et al.*, "Dynamic buffer allocation for thermal-aware 3d network-on-chip systems," in *ICCE-TW*, 2017.

[24] M.-H. Haghbayan *et al.*, "MapPro: Proactive Runtime Mapping for Dynamic Workloads by Quantifying Ripple Effect of Applications on Networks-on-Chip," in *NOCS*, 2015.

[25] J. Hamers *et al.*, "Scenario-based resource prediction for qos-aware media processing," *Computer*, 2010.

[26] D. Lo *et al.*, "Towards energy proportionality for large-scale latency-critical workloads," in *ISCA*, 2014.

[27] C. Delimitrou *et al.*, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *ASPLOS*, 2013.

[28] M. P. Papazoglou *et al.*, "Introduction: Service-oriented computing," *Commun. ACM*, 2003.

[29] P. Ranganathan *et al.*, "Enterprise it trends and implications for architecture research," in *HPCA*, 2005.

[30] V. Petrucci *et al.*, "Octopus-man: Qos-driven task management for heterogeneous multicores in warehouse-scale computers," in *HPCA*, 2015.

[31] J. Mars *et al.*, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *MICRO*, 2011.

[32] H. Yang *et al.*, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *ISCA*, 2013.

[33] L. Tang *et al.*, "Compiling for niceness: Mitigating contention for qos in warehouse scale computers," in *CGO*, 2012.

[34] Y. Ding *et al.*, "Qos aware dynamic time-slice tuning," in *IISWC*, 2014.

[35] Y. Zhou *et al.*, "Cash: Supporting iaas customers with a sub-core configurable architecture," in *ISCA*, 2016.

[36] H. Sung *et al.*, "Ombm: Optimized memory bandwidth management for ensuring qos and high server utilization," in *FAS* W*, 2017.

[37] L. Tang *et al.*, "The impact of memory subsystem resource sharing on datacenter applications," in *ACM SIGARCH Computer Architecture News*. ACM, 2011.

[38] L. Subramanian *et al.*, "The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory," in *MICRO*, 2015.

[39] H. Kasture *et al.*, "Ubik: Efficient cache sharing with strict qos for latency-critical workloads," in *ASPLOS*, 2014.

[40] R. Iyer *et al.*, "Qos policies and architecture for cache/memory in cmp platforms," in *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2007.

[41] A. Herdrich *et al.*, "Cache qos: From concept to reality in the intel® xeon® processor e5-2600 v3 product family," in *HPCA*, 2016.

[42] A. Sharifi *et al.*, "Mete: Meeting end-to-end qos in multicores through system-wide resource management," *SIGMETRICS*, 2011.

[43] N. Beckmann *et al.*, "Scaling distributed cache hierarchies through computation and data co-scheduling," in *HPCA*, 2015.

[44] Y. Ye *et al.*, "Coloris: A dynamic cache partitioning system using page coloring," in *PACT*, 2014.

[45] X. Zhang *et al.*, "Towards practical page coloring-based multicore cache management," in *EuroSys*, 2009.

[46] B. Li *et al.*, "Dynamic qos management for chip multiprocessors," *ACM Trans. Archit. Code Optim.*, 2012.

[47] B. Li *et al.*, "Coqos: Coordinating qos-aware shared resources in noc-based socs," *Journal of Parallel and Distributed Computing*, 2011.

[48] B. Grot *et al.*, "Kilo-noc: A heterogeneous network-on-chip architecture for scalability and service guarantees," in *ISCA*, 2011.

[49] J. W. Lee *et al.*, "Globally-synchronized frames for guaranteed quality-of-service in on-chip networks," in *ISCA*, 2008.

[50] J. Ouyang *et al.*, "Loft: A high performance network-on-chip providing quality-of-service support," in *MICRO*, 2010.

[51] J. Diemer *et al.*, "Back suction: Service guarantees for latency-sensitive on-chip networks," in *NOCS*, 2010.

[52] J. Diemer *et al.*, "Efficient throughput-guarantees for latency-sensitive networks-on-chip," in *ASP-DAC*, 2010.

[53] B. Grot *et al.*, "Preemptive venergy efficient proactive thermal mirtual clock: A flexible, efficient, and cost-effective qos scheme for networks-on-chip," in *MICRO*, 2009.

[54] J. Heißwolf *et al.*, "A scalable noc router design providing qos support using weighted round robin scheduling," in *ISPA*, 2012.

[55] S. Misailovic *et al.*, "Quality of service profiling," in *ICSE*, 2010.

[56] M. A. Laurenzano *et al.*, "Input resonsiveness: Using canary inputs to dynamically steer approximation," in *PLDI*, 2016.

[57] M. Samadi *et al.*, "Sage: Self-tuning approximation for graphics engines," in *MICRO*, 2013.

[58] S. Sidiroglou *et al.*, "Managing performance vs. accuracy trade-offs with loop perforation," in *SIGSOFT/FSE*, 2011.

[59] W. Baek *et al.*, "Green : A Framework for Supporting Energy-Conscious Programming using Controlled Approximation," in *PLDI*, 2010.

[60] H. Esmaeilzadeh *et al.*, "Neural Acceleration for General-Purpose Approximate Programs," in *MICRO*, 2012.

[61] T. Wang *et al.*, "Approxqa: a unified quality assurance framework for approximate computing," in *DATE*, 2017.

[62] C. Xu *et al.*, "On quality trade-off control for approximate computing using iterative training," in *DAC*, 2017.

[63] B. Grigorian *et al.*, "Brainiac: Bringing reliable accuracy into neurally-implemented approximate computing," in *HPCA*, 2015.

[64] C. Xu *et al.*, "On quality trade-off control for approximate computing using iterative training," in *DAC*, 2017.

[65] T. Moreau *et al.*, "Exploiting quality-energy tradeoffs with arbitrary quantization: special session paper," in *CODES+ISSS*, 2017.

[66] X. Sui *et al.*, "Proactive control of approximate programs," in *ASPLOS*, 2016.

[67] D. S. Khudia *et al.*, "Rumba: An online quality management system for approximate computing," in *ISCA*, 2015.

[68] D. Mahajan *et al.*, "Towards statistical guarantees in controlling quality tradeoffs for approximate acceleration," in *ISCA*, 2016.

[69] Semiconductor Industry Association *et al.*, "International Technology Roadmap for Semiconductors," http://www.itrs2.net/, 2011.

[70] JEDEC Solid State Tech. Ass., "Failure mechanisms and models for semiconductor devices," *JEDEC Publication JEP122G*, 2010.

[71] E. Karl *et al.*, "Multi-Mechanism Reliability Modeling and Management in Dynamic Systems," *TVLSI*, 2008.

[72] J. Srinivasan *et al.*, "The Case for Lifetime Reliability-Aware Microprocessors," in *ISCA*, 2004.

[73] W. J. Song *et al.*, "Managing performance-reliability tradeoffs in multicore processors," in *IRPS*, 2015.

[74] A. Coskun *et al.*, "Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors," in *Proc. Int. Conf. Measurement and Modeling of Computer Systems*, 2009.

[75] K. Ma *et al.*, "PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs," in *PACT*, 2012.

[76] M. H. Haghbayan *et al.*, "A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era," in *DATE*, 2016.

[77] Y. Ma *et al.*, "Improving System-Level Lifetime Reliability of Multicore Soft Real-Time Systems," *IEEE TVLSI*, 2017.

[78] J. Sun *et al.*, "Workload Assignment Considering NBTI Degradation in Multicore Systems," *Journal Emerg. Technol. Comput. Syst.*, 2014.

[79] T. Kim *et al.*, "Learning-based dynamic reliability management for dark silicon processor considering EM effects," in *DATE*, 2016.

[80] A. Baldassari *et al.*, "A dynamic reliability management framework for heterogeneous multicore systems," in *DFT*, 2017.

[81] X. Chen *et al.*, "Run-time technique for simultaneous aging and power optimization in GPGPUs," in *DAC*, 2014.

[82] H. Lee *et al.*, "Aging-aware Workload Management on Embedded GPU Under Process Variation," *IEEE Trans. on Computers*, 2018.

[83] M. H. Haghbayan *et al.*, "Performance/Reliability-Aware Resource Management for Many-Cores in Dark Silicon Era," *IEEE Trans. on Computers*, 2017.

[84] N. Kapadia *et al.*, "Varsha: Variation and reliability-aware application scheduling with adaptive parallelism in the dark-silicon era," in *DATE*, 2015.

[85] Y. Xiang *et al.*, "Soft and hard reliability-aware scheduling for multicore embedded systems with energy harvesting," *IEEE TMSCS*, 2015.

[86] Y. Ma *et al.*, "An on-line framework for improving reliability of real-time systems on x201c;big-little x201d; type mpsocs," in *DATE*, 2017.

[87] C. Bolchini *et al.*, "Self-adaptive fault tolerance in multi-/many-core systems," *Journal of Electronic Testing*, 2013.

[88] M. H. Haghbayan *et al.*, "A power-aware approach for online test scheduling in many-core architectures," *IEEE Transactions on Computers*, 2016.

[89] M. A. Skitsas *et al.*, "Exploring system availability during software-based self-testing of multi-core cpus," *Journal of Electronic Testing*, 2018.

[90] R. Bitirgen *et al.*, "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in *MICRO*, 2008.

[91] C. Dubach *et al.*, "Dynamic Microarchitectural Adaptation Using Machine Learning," *ACM TACO*, 2013.

[92] E. Ipek *et al.*, "Self-optimizing memory controllers: A reinforcement learning approach," in *ISCA*, 2008.

[93] Y. Zhang *et al.*, "Intelligent cloud resource management with deep reinforcement learning," *IEEE Cloud Computing*, 2017.

[94] N. Mishra *et al.*, "Caloree: Learning control for predictable latency and low energy," in *ASPLOS*, 2018.

[95] H. Hoffmann *et al.*, "Dynamic Knobs for Responsive Power-aware Computing," in *ASPLOS*, 2011.

[96] A. M. Rahmani et al., "Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach," in *ISLPED*, 2015.

[97] A. M. Rahmani *et al.*, "Reliability-Aware Runtime Power Management for Many-Core Systems in the Dark Silicon Era," *IEEE TVLSI*, 2017.

[98] A. Kanduri *et al.*, "Approximation knob: Power Capping meets energy efficiency," in *ICCAD, 2016*.

[99] M. H. Haghbayan *et al.*, "Performance/Reliability-Aware Resource Management for Many-Cores in Dark Silicon Era," *IEEE Tran. on Computers*, 2017.

[100] A. Kanduri *et al.*, "adBoost: Thermal Aware Performance Boosting through Dark Silicon Patterning," *IEEE Tran. on Computers*, 2018.

[101] A. M. Rahmani *et al.*, "Reliability-aware runtime power management for many-core systems in the dark silicon era," *IEEE TVLSI*, 2017.

[102] J. L. Hellerstein *et al.*, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[103] S. Shahosseini *et al.*, "Dependability evaluation of siso control-theoretic power managers for processor architectures," in *NORCAS*, 2017.

[104] R. P. Pothukuchi *et al.*, "Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures," in *ISCA*, 2016.

[105] T. R. Muck *et al.*, "Design methodology for responsive and robust mimo control of heterogeneous multicores," *IEEE TMSCS*, 2018.

[106] C. Karamanolis *et al.*, "Designing Controllable Computer Systems," in *HOTOS*, 2005.

[107] P. Ramadge *et al.*, "The control of discrete event systems," *Proceedings of the IEEE*, 1989.

[108] "Supervisory control of discrete event systems," *Mathematical and Computer Modelling*, 1996.

[109] A. M. Rahmani *et al.*, "Spectr: Formal supervisory control and coordination for many-core systems resource management," in *ASPLOS*, 2018.