THE SERVICES COORDINATOR: ORCHESTRATING THE BEHAVIOR OF INDEPENDENT ADAPTIVE SYSTEMS

Jacopo Panerati [†], *Martina Maggio* [‡], *Matteo Carminati* [†], *Marco Triverio* [†], *Marco D. Santambrogio* [†]
[†] Dipartimento di Elettronica e Informazione, Politecnico di Milano
[‡] Automatic Control, Lund University
{jacopo.panerati,marco.triverio}@mail.polimi.it, martina.maggio@control.lth.se
{mcarminati,santambrogio}@elet.polimi.it

ABSTRACT

Nowadays the complexity of computing systems is skyrocketing. Programmers have to deal with extremely powerful computing systems that take time and considerable skills to be instructed to perform at their best. This work analyzes the stated problem and proposes a simple, yet powerful mechanism for optimizing performance through the coordination of the interaction of multiple, independent adaptive systems called *services*. In this scenario we developed the Services Coordinator, a system-centralized *decision engine* based on reinforcement learning. The Services Coordinator gathers information about the performance goals of the system and can either turn services *on* or *off*. The Services Coordinator analyzes the runtime impact of services and of their autonomous decision policies, looking for a combination of services that makes it possible to reach the given goals. The experiments that have been carried out show the ability of the Services Coordinator to adapt to changing conditions, confirming the validity and the flexibility of the followed approach.

1. INTRODUCTION

The power and the complexity of computing systems are evolving and increasing at an unprecedented rate. On one hand, the advantages of highly-parallel systems could benefit an enormous variety of fields. On the other hand, the growing complexity is making it unfeasible for the average programmer to weight all the constraints and optimize the system for a wide range of machines and scenarios [1]. Even though technologies have improved, making a system perform at its best is a non-trivial task. The burden on programmers is noticeable and many research efforts were spent in addressing this issue. Clearly, it is not feasible to rely on human intervention to tune a system: conditions change constantly, rapidly, and unpredictably. It would be desirable to have the system automatically *adapt* to the mutating environment [2].

A commonly shared opinion is the need for new paradigms to be explored and for new frameworks to be developed. Among those, self-adaptive systems seem to be the answer to most of the problems previously described [2]. Self-Aware Adaptive computing systems adapt behavior and resources to automatically find the best way to accomplish a given goal despite changing environmental conditions and demands. Therefore, this kind of system needs to monitor itself and its context, discern significant changes, determine how to react, and execute decisions.

This work presents a solution able to adjust itself during execution due to a simple, yet powerful mechanism for coordinating the interaction of multiple, existing adaptive systems, each of them with its own decision engine. The system used in this paper considers current computing and operating systems augmented through the usage of a modular ecosystems of software components called *services*, capable of actuating a change on the system. The proposed solution can be used within different architectures, from mobile devices (e.g. mobile phones), to desktops, servers and laptops. In this scenario we developed a system-centralized *decision engine*, called Services Coordinator and based on reinforcement learning. The Services Coordinator gathers all the information about system performance and is able to decide which measures to enact, orchestrating the different elements that commit changes to the environment. The Services Coordinator is fully integrated in a standard Linux operating system, aiming at making it a Self-Aware Adaptive computing system, and co-exists with other independent and adaptive decision making entities, which reside in services. The Services Coordinator turns services either *on* or *off* as necessary to meet the performance goals of the system. We validated the framework with preliminary tests, aimed at evaluating the approach and the feasibility of the solution.

The remainder of this paper is organized as follows. Section 2 presents the context of adaptive operating systems. Section 3 introduces and describes in detail the proposed methodology and the machine learning-based decision engine. In Section 4 our experimental results are presented. Finally, Section 5 concludes the paper.

2. CONTEXT DEFINITION

The goal of the Services Coordinator is to enhance the Operating System (OS) with a novel adaptivity layer at run time. A common Linux distribution is chosen here as the target OS to show the wide applicability of the approach in both designing a completely new OS and complementing existing code. In literature, some projects exist that are explicitly trying to enable self-awareness and adaptiveness in a newly designed OS. Space limitations allow only a brief description of main contributions in the area: *K42* [3] and *Sefos* [4].

K42 is a research kernel designed for cache-coherent 64-bit multiprocessor and NUMA systems. Among its declared goals there is allowing applications to customize the OS behavior in how the OS is managing the resources devoted to them. Another stressed goal is letting the system adapt to changing workload characteristics. To achieve these goals, the overall structure of K42 is based on the object-oriented paradigm and on a modular design based on microkernels. K42 proposes a layered architecture based on kernels, servers, and user-level libraries for applications developing. The servers marshal all the operating system functionalities, therefore introducing adaptation at the OS level means simply modifying the behavior of these servers. It is noticeable that however, a central, coordinating entity is not included in this structure.

SElf-aware Factored Operating System (Sefos) is a self-aware OS specifically designed for scalability on many-cores architectures. It is based on the *fos* [5] operating system, and integrated with *SEEC* [6] (SElf-awarE Computational model), the purpose of the integration being the introduction of the self-adaptation layer. To do this, a typical decision loop is implemented: the autonomic system executes and monitors itself using sensors. The system is able to react to the sensed conditions, taking decisions and acting to guarantee application performance. The adopted monitoring interface is Application Heartbeats [7, 8]. There is a decision engine acting on the system to set the values of each decision parameter. However, when multiple decisions are taken at the same time, it is not clear how the coordination between the different actuation mechanisms available in the system takes place. Notice also that Sefos relies on trusted actuators: no security checks are performed on the taken decision, thus malicious entities cannot be detected and deactivated.

Similarly to Sefos, the adaptive system the Services Coordinator lives in, implements a decision loop. However the Services Coordinator represents a new entity: it is a system-centralized decision engine, learning how to enable and disable the available adaptation mechanisms in order to meet the high level goals of the system.

3. PROPOSED METHODOLOGY

To better understand the importance and the rationale behind the Services Coordinator, the general scenario[1] that we are addressing is briefly presented. It is made of applications, processes and monitored processes, services and the Services Coordinator, in a system structured as shown in Figure 1:

- *Applications* – Pieces of software written to accomplish a specific task.
- *Processes* – Instances of an application.
- *Monitored Processes* – Processes that are making one or more entities of the system aware of their performance goals and actual progresses. To do this, the Application Heartbeats framework [8, 7] is used.
- *Services* – A service represents a component capable of performing changes on one or more applications or on the whole system. Services are enabled or disabled by our decision engine; when enabled, they are autonomous components that can decide and act on the system, for example reading the performance signals of the applications and/or computing some reaction to changes in the external environment.
- *Services Coordinator* – It is the object the focus of this document is on. It is aware of monitored processes and services, gathers data about applications performance levels and acts enabling or disabling services. Its action policy is based on a machine-learning technique.

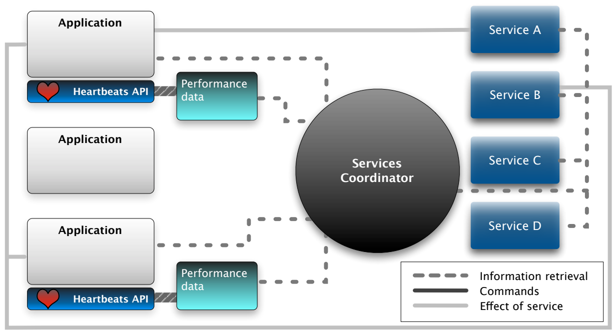

Fig. 1. The Services Coordinator architecture.

In this context, the applications set their performance goals and the *Services Coordinator* orchestrates the available services boosting (or reducing) performance in order to make it possible to achieve the given goals. This scenario features the Services Coordinator as a central element that has a global vision of the system.

We believe this approach has the power to achieve, if possible, a global optimum. We are conscious of the theoretical fragility of the proposed architecture, the Services Coordinator representing a *single point of failure*. This condition has been partially mitigated by our implementation, which allows each enabled component of the framework to

[1]The description of the overall approach with details of each component, presenting a set of experiments to prove its effectiveness, is out of the scope of this paper. This paper is focusing its attention of the Services Coordinator, the key component at the center of the decision loop.

run even in case of failure of one or more other components. However, the research is still ongoing on this topic and more experiments have to be carried out to evaluate different solutions.

3.1. Orchestrating the services

The Services Coordinator stands at the center of the decision loop and exploits the awareness given by the available observation mechanisms (in this case Application Heartbeats) to elaborate a plan for future behavior. The aim is to tune performance in order to make each monitored process achieve its performance goals. In particular, the Services Coordinator exploits the information coming from the monitor in order to determine the available services and the monitored applications; constantly verifies the availability of new or old services and the presence of new or old monitored processes; gathers the information coming from the monitored processes; analyzes the performance-related data in order to understand whether to enact a correction policy; decides which services to enable or disable and communicates them these decisions.

The decision policy that drives the Services Coordinator is based on *machine learning* techniques, which were proved powerful tools for managing the increasing complexity of computing systems [9, 10, 7]. We implemented *R-learning*, a *reinforcement learning* algorithm [11]. In general, Reinforcement Learning augments a system with the possibility to learn from experience through the use of a *reward signal* that drives the learning process. In particular, the algorithm calculates a signal that is a synthesis of the current state (and of its performance characteristics, such as whether the applications are in the desired performance range); it then selects actions attempting to maximize the given reward. A Reinforcement Learning technique is needed since the Services Coordinator has no a-priori knowledge about the action performed by each service in the system. Within this context, it has to discover the effects of each independent adaptive system enabling and disabling it. Furthermore, Reinforcement Learning provides an efficient method to build knowledge from experience.

Once a strategy has been decided by the Services Coordinator, it must be enacted. In our frameworks the actuators are called *services*. There are many kinds of services that might be available and that might affect the system in different ways: some might affect performance, accuracy, or both; some might impact on the whole system while others might target one or more applications. In the following we focus on a couple of services but many others could be envisioned (e.g., lock mechanisms [7], memory allocation and frequency scaling [6]). The two services we address are core allocation and priority adjusting. Both these optimizations are enabled on a per-application basis. It is worth stressing that this means that a service per application could be activated.

In our design of the Services Coordinator we paid attention to some of its characteristics. First, the interface between the Services Coordinator and the services is extremely simple. Second, there is no need to model a service – something that might prove truly difficult given the heterogeneity of the computing systems on which the framework could run. Moreover, the Services Coordinator does not need to be updated or restarted when new services are plugged in, to the extent of self-configuration. All services embed a decision making mechanism that is independent from the Services Coordinator: the Services Coordinator only enables or disables services. When a conscious service is enabled its own loop-based decision mechanisms is activated.

4. PRELIMINARY RESULTS

The experimental evaluation was able to enable/disable a multi-application version of the *core-allocator* presented in [6] and a newly developed service named *priority adjuster*. Note that the proposed methodology is general and may be applied with any other service that implements the Services API. The monitored processes were instances of x264, possibly with different parameters, a different number of threads, and different requirements. In this scenario, x264 was run to see its behavior during an uncontrolled execution (neither the Services Coordinator or any of the services are running). The heart rate is bound between 80 and 90 heartbeats per second. On average, with the given parameters and the given number of cores, it signals 85.49 heartbeats per second.

4.1. Overhead

Previous research has shown that the Application Heartbeats framework has a very limited overhead (e.g., only circa 4% on an application encrypting and decrypting through the DES algorithm). We have then analyzed the Services Coordinator and the *Services Application Programming Interface API* in order to quantify the overhead of the framework. In particular, we run an Application Heartbeats instrumented version of x264[2] without the Services Coordinator and with the Services Coordinator and the *core allocator* on an x86-64 Intel Core i7-870 processor[3] and compared the results. The same experiment has been repeated 10 times and then averaged. The Services Coordinator-enabled version has an overhead of circa 5%, an encouraging result.

4.2. Approach validation

In this subsection we present different tests, carried out on the same machine used for the overhead tests. The experimental evaluation was able to enable/disable a multi-application version of the *core-allocator* presented in [6] and a newly developed service named *priority adjuster*. Note that the

[2]x264 is an open-source application for encoding video streams into the H.264/MPEG-4 AVC format.

[3]Clock speed: 2.93 GHz, 4 GB of SDRAM DDR3-1333, NVIDIA GeForce GT 240 graphic card, OS: Ubuntu 9.10.

proposed methodology is general and may be applied with any other service that implements the Services API. The monitored processes were instances of x264, possibly with different parameters, a different number of threads, and different requirements. In this scenario, x264 was run to see its behavior during an uncontrolled execution (neither the Services Coordinator or any of the services are running). The heart rate is bound between 80 and 90 heartbeats per second. On average, with the given parameters and the given number of cores, it signals 85.49 heartbeats per second.

4.3. Different performance goals

This experiment tests the reaction of the system in a multi-application and multi-service domain: a second application is started around 30 seconds from the beginning of the experiment. Both the applications are instances of x264, with different performance goals (desired heart rate of 20 to 30 heartbeats per second for the first application and 30 to 50 for the second one). The system behavior is shown in Figure

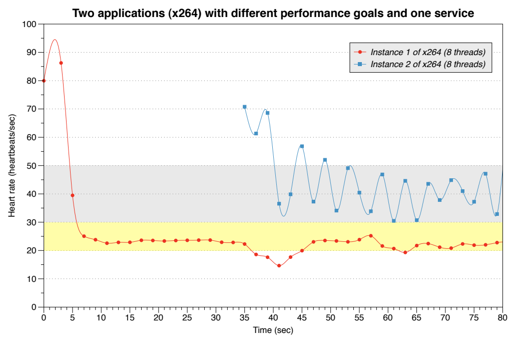

Fig. 2. Two instances of x264 with different performance goals. Desired heart rate ranges are in yellow and gray.

2. When the second x264 instance is introduced the Services Coordinator explores the new state-action space entering a suboptimal condition for the first application. It is however able to recognize the best action to be taken almost immediately, allowing both instances to meet their goals.

4.4. Stress test

This test has been run with eight x264 monitored instances, each of them trying to achieve the same performance goals, their heart rate being between 5 and 10 heartbeats per second. In this experiment eight services can be activated and deactivated, each one being the core allocator for a specific x264 process. Figure 3 shows the results for this test, where the solution space is made up of 256 possible configurations. The time needed to learn the system reactions is intuitively higher than in the previous experiments; nevertheless this test confirms the flexibility of the algorithm. Moreover, the Services Coordinator reaction is fast enough to drive all the eight applications to their desired performance level.

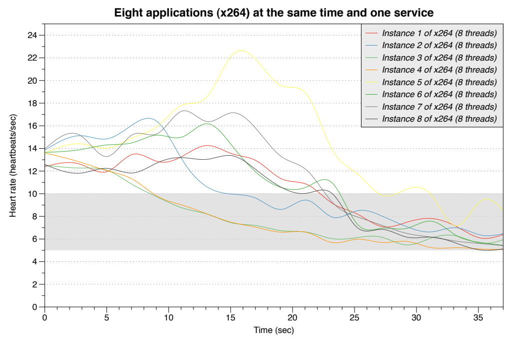

Fig. 3. Eight instances of x264.

5. CONCLUSIONS

In this paper we presented a component, called Services Coordinator, that acts as a decision engine in a self-aware architecture capable of performing optimizations on itself, and adapting to unpredictable, unknown, and unfavorable conditions. The Services Coordinator is the central entity that gathers all the information coming from applications and decides which actions to perform in order to make the processes reach the desired goals. It implements a decision algorithm called *R-learning*, allowing the component to learn from experience and to optimize the performance of the whole system. The introduction of such central entity, based on machine learning, represents the prior contribution of this paper with respect to the works described in literature. Without any prior information on the services, it is shown that the Services Coordinator is able to learn and dynamically improve the quality of the system.

6. REFERENCES

[1] P. Horn, "Autonomic computing: Ibm perspective on the state of information technology," in *IBM Germany Scientific Symposium Series*, 2001.

[2] M. Salehie *et al.*, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 1–42, 2009.

[3] J. Appavoo, *et al.*, "K42 Overview," http://www.research.ibm.com/K42/white-papers/Overview.pdf, August 2002.

[4] Various Authors, "The mit angstrom project: Universal technologies for exascale computing (project home page)," http://projects.csail.mit.edu/angstrom/index.html.

[5] D. Wentzlaff, *et al.*, "An operating system for multicore and clouds: mechanisms and implementation," in *SoCC*, 2010, pp. 3–14.

[6] H. Hoffmann, *et al.*, "Seec: A framework for self-aware computing," MIT CSAIL, Tech. Rep., October 2010.

[7] J. Eastep, *et al.*, "Smartlocks: lock acquisition scheduling for self-aware synchronization," in *7th international conference on Autonomic computing*, ser. ICAC '10. New York, NY, USA: ACM, 2010, pp. 215–224. [Online]. Available: http://doi.acm.org/10.1145/1809049.1809079

[8] H. Hoffmann, *et al.*, "Application heartbeats for software performance and health," *15th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pp. 347–348, January 2010.

[9] E. Ipek, *et al.*, "Self-optimizing memory controllers: A reinforcement learning approach," in *35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 39–50. [Online]. Available: http://dx.doi.org/10.1109/ISCA.2008.21

[10] R. Bitirgen, *et al.*, "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in *41st annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 41. Washington, DC, USA: IEEE Computer Society, 2008, pp. 318–329. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2008.4771801

[11] R. S. Sutton *et al.*, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998.