# Chapter 5
# Low-Overhead Development of Scalable Resource-Efficient Software Systems

**Wei-Chih Huang**
*Imperial College London, UK*

**William Knottenbelt**
*Imperial College London, UK*

## ABSTRACT

*As the variety of execution environments and application contexts increases exponentially, modern software is often repeatedly refactored to meet ever-changing non-functional requirements. Although programmer effort can be reduced through the use of standardised libraries, software adjustment for scalability, reliability, and performance remains a time-consuming and manual job that requires high levels of expertise. Previous research has proposed three broad classes of techniques to overcome these difficulties in specific application domains: probabilistic techniques, out of core storage, and parallelism. However, due to limited cross-pollination of knowledge between domains, the same or very similar techniques have been reinvented all over again, and the application of techniques still requires manual effort. This chapter introduces the vision of self-adaptive scalable resource-efficient software that is able to reconfigure itself with little other than programmer-specified Service-Level Objectives and a description of the resource constraints of the current execution environment. The approach is designed to be low-overhead from the programmer's perspective – indeed a naïve implementation should suffice. To illustrate the vision, the authors have implemented in C++ a prototype library of self-adaptive containers, which dynamically adjust themselves to meet non-functional requirements at run time and which automatically deploy mitigating techniques when resource limits are reached. The authors describe the architecture of the library and the functionality of each component, as well as the process of self-adaptation. They explore the potential of the library in the context of a case study, which shows that the library can allow a naïve program to accept large-scale input and become resource-aware with very little programmer overhead.*

DOI: 10.4018/978-1-4666-6026-7.ch005

## 1. INTRODUCTION

Modern software engineers are faced with an explosion in the number of execution environments in which their applications might execute (e.g. smartphone, tablet, laptop, server, etc.). In each of these potential execution environments, each class of application is subject to different resource constraints and may also be subject to different Quality of Service (QoS) requirements. Consider Figure 1, which presents the importance of three common QoS parameters (performance, memory efficiency, and reliability) in different application contexts and execution environments. For example, when a game is operated on a game console, the game's performance should be at a very high level to meet players' expectations, possibly leading to the use of more memory space and higher electric power consumption. By contrast, if the game is executed on a smartphone, lower performance may be tolerated to save battery power and to use less memory space. Similarly, if a web browser runs on a smartphone, due to high usage frequency of the web browser and limited memory space of the smartphone, high performance with low memory consumption is expected. But when the web browser is executed on a server or a game console, the demand of high performance with low memory consumption is not required because these two platforms can provide sufficient memory space and are not frequently used to surf the internet.

It is a major challenge to write software capable of maintaining QoS in every possible execution environment and application context, especially in the face of bursty and/or high-intensity workloads that may frequently stretch or exceed resource limitations. To avoid unacceptable degradations in the quality of user experience, it is necessary to implement mechanisms for scalability, robustness and intelligent resource exploitation. Even if sound software engineering principles are applied to maximise software reuse, there are major barriers to the application of traditional software development techniques in light of these challenges. Specifically, significant manual reimplementation and refactoring must be carried out for each execution environment, and substantial levels of programmer expertise is necessary.

To address this situation, we propose a self-adaptive framework for "intelligent" software which adapts at run-time to the resource constraints of its present execution environment, as well as automatically scaling up to handle large input sizes,

*Figure 1. The importance of QoS requirements on different application contexts and execution environments*