Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Reducing resource consumption of expandable collections: The Pharo case

Alexandre Bergel [a,*], Alejandro Infante [a], Sergio Maass [a], Juan Pablo Sandoval Alcocer [a,b]

[a] Pleiad Lab, DCC, University of Chile, Chile
[b] Universidad Mayor de San Simón, Bolivia

ARTICLE INFO

Article history:
Received 30 January 2017
Received in revised form 11 December 2017
Accepted 19 December 2017
Available online xxxx

Keywords:
Collection
Pharo
Lua
Profiling
Experiment

ABSTRACT

Expandable collections are collections whose size may vary as elements are added and removed. Hash maps and ordered collections are popular expandable collections. Expandable collection classes offer an easy-to-use API, however this apparent simplicity is accompanied by a significant amount of wasted resources.

We describe some improvements of the collection library to reduce the amount of waste associated with collection expansions. We have designed two new collection libraries for the Pharo programming language that exhibit better resource management than the standard library. We improved the Pharo collection library using two complementary perspectives.

First, across a basket of 5 applications, our optimized collection library significantly reduces the memory footprint of the collections: (i) the amount of intermediary internal array storage by 73%, (ii) the number of allocated bytes by 67% and (iii) the number of unused bytes by 72%. This reduction of memory is accompanied by a speedup of about 3% for most of our benchmarks.

Second, we looked for an alternative to the classical expandable collection. The Lua programming language offers a unique abstract data type called *table*. We designed, implemented, and introduced this data type in the Pharo programming language and we ran a number of micro and macro-benchmarks. Overall, replacing the standard Pharo collection library by one inspired on Lua's table data type results in an execution speedup of up to 15% and a reduction of the memory consumption by up to 19%.

We analyzed the collection implementations of Java, C#, Scala, and Ruby: these implementations largely behave like Pharo's, therefore with the same limitations. Our results are thus likely to benefit designers of future programming languages and collection libraries.

1. Introduction

Creating and manipulating any arbitrary group of values is largely supported by today's programming languages and runtimes [1]. A programming environment typically offers a collection library that supports a large range of variations in the way collections of values are handled and manipulated. Collections exhibit a wide range of features [1–3], including

* Corresponding author.
E-mail address: abergel@dcc.uchile.cl (A. Bergel).

https://doi.org/10.1016/j.scico.2017.12.009
0167-6423/

being expandable or not. An expandable collection is a collection whose size may vary as elements are added and removed. Expandable collections are highly popular among practitioners and have been the topic of a number of studies [4–7].

Expandable collections are typically implemented by wrapping a fixed-sized array. An operation on the collection is then translated into primitive operations on the array, such as copying the array, replacing the array with a larger one, inserting or removing a value at a given index.

Unfortunately, the simplicity of using expandable collections is counter-balanced by resource consumption when not adequately employed [4,5,8]. Consider the case of a simple ordered collection (e.g., ArrayList in Java and OrderedCollection in Pharo). Using the default constructor, the collection is created empty with an initial capacity of 10 elements. The 11th element added to it triggers an expansion of the collection by doubling its capacity. This brief description summarizes the behavior of most of the expandable collections in Java, C#, Scala, Ruby, and Pharo.

We have empirically determined that in Pharo a large portion of collections created by applications are empty. As a consequence, their internal arrays are simply unused. Moreover, only a portion of the internal array is used. After adding 11 elements to an ordered collection, 9 of the 20 slot arrays are left unused. Situations such as this one scale up as soon as millions of collections are involved in a computation.

We have selected the Pharo programming language for our study. Pharo[1] is an object-oriented dynamically typed programming language which offers a large and rich collection library [9]. Pharo is syntactically close to Ruby and Objective-C. Conducting our experiment in Pharo has a number of benefits. Firstly, Pharo offers an expressive reflective API which greatly reduces the engineering effort necessary to modify and replace the collection library. Secondly, the open source community that supports Pharo is friendly and is looking for contributions for improvement, which means that our results are to have a measurable impact across Pharo developers. In principle, our technique may be implemented in an highly optimized environment such as Java or .Net. However, we avoided a statically typed language for two reasons: (i) the runtime and the JIT depend on the Collection library,[2] as such our measurements would measure the implementation of the JIT, which would radically change the focus of the article; (ii) it is unclear whether Lua's tables may be implemented in a statically typed language (again, studying this question would change the focus on the present article). Appendix A briefly presents the syntax of Pharo.

This article is about measuring wasted resources in Pharo (memory and execution time) due to expandable collections. Improvements are then deduced and we measure their impact. We made two improvements to the Pharo collection library.

We improve first the way Pharo collection classes behave by using popular techniques: lazy object creation and recycling objects in a pool of frequently created objects. This article carefully evaluates the application of these well known techniques on the collection implementation. The analyses that this article describes focus on the profiling of over 6 million expandable collections produced by 15 different program executions. Note that our intent is not to prohibit a manual setting of the expansion strategy. Instead, we provide a simple mechanism to complement existing expandable collections.

Second, we looked for an alternative schema of the classical way expandable collections are implemented. Lua is a popular programming language that offers tables, a hybrid abstract data type combining features of sequential collections and dictionaries. We describe tables in Lua and compare their performance with the dictionary and the sequential ordered collection. Our experiments show that when replacing instances of the standard Pharo classes `OrderedCollection` and `Dictionary` by our implementation of Lua's tables, the memory allocation due to collections is decreased by up to 19% when executing long-running benchmarks that make extensive use of collections. It also has an impact on execution time, which is overall decreased, with a maximal reduction of 15%. Our novelty is about (i) porting tables to a language that contains a different library framework, (ii) adapting code using the Pharo standard library to use tables, and (iii) measuring the gain in terms of resource consumption.

The research questions we are pursuing are:

A – *How to characterize the use of expandable collections in Pharo?* Understanding how expandable collections are used is highly important in identifying whether or not some resources are wasted. And if this is case, how such waste occurs.

B – *Can the overhead associated with expandable collections in Pharo be measured?* Assuming the characterization of collection expansions revealed some waste of resources, measuring such waste is essential to properly benchmark improvements that are carried out either on the application or the collection library.

C – *Can the overhead associated with expandable collections in Pharo be reduced?* Assuming that a benchmark to measure resource waste has been established, this question focuses on whether the resource waste accompanying the use of a collection library can be reduced without disrupting programmer habits.

Our results show the Pharo collection library can be significantly improved by (i) considering lazy array creation and recycling those arrays, and (ii) by using hybrid collections (OrderedCollection and Dictionary). The expandable collections of Java, Scala, Ruby and C# are very similar to those of Pharo, and therefore largely exhibit the same deficiencies, as described in Section 10. We therefore expect our recommendations to be beneficial to these languages as well.

[1] http://www.pharo-project.org.

[2] https://stackoverflow.com/questions/33317720/performance-of-collections-emptylist-and-empty-arraylist-with-jit-compiler.

Please cite this article in press as: A. Bergel et al., Reducing resource consumption of expandable collections: The Pharo case, Sci. Comput. Program. (2017), https://doi.org/10.1016/j.scico.2017.12.009