Moving Java to Big Data

Author Sebastiano Vigna

Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Italia

1 The premise

`fastutil`, the DSI Utilities, Sux4J, MG4J, WebGraph and the **LAW** software constitute a family of software component for the management of very large sets of data, in particular documents collections and web graphs coming from web crawls.

However, "very large" is relative. Intrinsic limitations of the Java language (indices of arrays and lists must be 32-bit `int`s; the size of a collection is returned as a 32-bit `int`) makes it very difficult to handle very large graphs, inverted indices, or even static data structure. For this reason, we decided to improve all the above software libraries so that a 64-bit `long` becomes the natural type indexing a collection. In particular, the high-performance compressed represention implemented by the class `BVGraph` now has a version supporting 64-bit nodes, and there is an MG4J version using 64 bits to index terms and pointers, and so on. This change requires a number of new classes, and in particular a strategy for simulating large arrays.

This document describes the various pieces that make the transition of our Java software to 64 bits possible. We believe that the strategies we opted for will be useful for other people in other projects.

2 General notes

The main idea is that a new namespace, `it.unimi.dsi.big`, will be used for incompatible implementations. For instance, `it.unimi.dsi.big.webgraph.BVGraph` is a 64-bit version of `it.unimi.dsi.webgraph.BVGraph`. The signatures of the two classes are radically different (e.g., because `numNodes()` now returns a `long`). Implementations for which there is no difference in signature have been simply internally modified to work with larger data.

This method makes it possible to keep backward compatibility, while allowing people to use the new classes without too much difficulty. We realize that having classes with the same name and similar packages but different signatures can lead to some confusion, but we expect code to use exclusively either standard or big classes.

There are a few exceptions, which will require sometimes recompilation and sometimes minor, cosmetic modification to code using our libraries:

- `it.unimi.dsi.util.LongBigList` (and the associated abstract implementation `it.unimi.dsi.util.AbstractLongBigList`) have been eliminated in favour of the (now standard) versions provided by `fastutil` (`it.unimi.dsi.fastutil.longs.LongBigList` and `it.unimi.dsi.fastutil.longs.AbstractLongBigList`, respectively). This will require, essentially, to change your import for these classes wherever they appear. Moreover, the new interface for big lists implements `it.unimi.dsi.fastutil.Size64`, and thus provides a `size64()` method to return the size as a 64-bit integer (the old interface used `length()` instead). Note that *implementations* (e.g., `EliasFanoMonotoneBigList`) provide a deprecated `length()` method to ease the transition, but the new interfaces have no such method. Depending on how you declared your variables, you might have to fix your calls to `length()`.
- The family of hashes based on Jenkins hash implemented in `Hashes` had a serious flaw (the results had the same high bits on short strings). The problem has been fixed, but the fix changed the value returned by the hash functions, which in turn forced to bump *all* serial-version identifiers of classes using those hashes. You should also check that you're not relying on these functions to return the same value as they did before (it is a quite rare evenience, but you never know).

Note that all libraries have been released in parallel: they work correctly when they are updated at the same time, but you will run into problems if you try to mix them. In particular, you need `fastutil` 6.3 or newer, the DSI Utilities 2.0 or newer, Sux4J 3.0 or newer, MG4J 4.0 or newer, WebGraph 3.0 or newer and **LAW** software 2.0 or newer.

The low-lev

3 `fastutil`

`fastutil` contains since version 6 (the jar, for the occasion, has been renamed `fastutil.jar`, so be sure to delete old copies of `fastutil5.jar`) an interface `Size64` that implements a deprecated `size()` method returning an `int` (as it happens for Java's `Collection`) and a `size64()` method returning a `long`. Note that by contract `size()` will return `Integer.MAX_VALUE` if the collection is larger than that, which is, of course, a disaster. Classes that want to circumvent this behaviour are invited to implement `Size64`. We remark that it is always a good idea to implement both `size()` *and* `size64()`, as the former might be implemented by a superclass in an incompatible way. If you implement `size64()`, just implement `size()` as a *deprecated* method returning `Math.min(Integer.MAX_VALUE, size64())`.

Users of very large collections can use an `instanceof` test to see whether a collection implements `size64()`. Interfaces that are inherenently thought for very large collections (e.g., the new `StrigMap` or `BigList` interfaces) implement `Size64` directly.

`fastutil` provides facilities for handling *big arrays*, which are simply arrays of arrays subject to some length condition. The class `BigArrays` and the corresponding type-specific versions (e.g., `IntBigArrays`) provide a wealth of static methods that do things with big arrays, including big versions of binary search, sorting, and so on. Correspondingly, the interface `BigList` provides list access using 64-bit indices; unfortunately, it was impossible to make this interface compatible with `List`, but some adapters are provided in the usual static container classes. Please read carefully the Javadoc documentation of `BigArrays` and `BigList` to understand the rationale behind those classes. We will make other big implementations available (e.g., big sets).

4 The DSI Utilities

The main improvement is the creation of big versions of all interfaces and implementations related to `StringMap`. The interface `it.unimi.dsi.big.util.StringMap` now is able to represent very large collections of strings, and its methods have been changed accordingly. `PrefixMap` will return `LongIntervals` instead of `Intervals`.

5 Sux4J

The changes in Sux4J, excluding to the shift from `it.unimi.dsi.util.LongBigList` to `it.unimi.dsi.fastutil.longs.LongBigList`, are all internals.

6 MG4J

Every class in MG4J has now a copy in the namespace `it.unimi.dsi.big` which uses 64-bit addressing for terms and documents. We expect the big version to become the standard version in which new developments are made. The internal format of indices has not changed, so this transition should be relatively smooth. Of course, maps representing string will have to be rebuilt.

An important change in the new setting is that the integer returned to represent an "end of list" condition can no longer be `Integer.MAX_VALUE`. A new constant `DocumentIterator.END_OF_LIST` has taken its place (it is actually `Long.MAX_VALUE`), and should replaced manually whenever `Integer.MAX_VALUE` has been used for that purpose.

7 WebGraph

Every class in Webgraph has now a copy in the namespace `it.unimi.dsi.big` which uses 64 bit for indexing a node. The number of nodes of a graph is now a `long`, and correspondingly all methods have been updated. Outdegrees are thus `long`s, as well as successors, and successor arrays are actually big arrays. Please read the WebGraph overview for more information.

Note that the WebGraph format has not changed, but the shift from `it.unimi.dsi.util.LongBigList` to `it.unimi.dsi.fastutil.longs.LongBigList` implies that the previously generated `.obl` files will be no longer loadable (just regenerate them).

Since WebGraph was already using a big list, `.obl` files can be interchangeably used in the standard and in the big version. The generic loading methods in `ImmutableGraph` will try to insert or replace the `.big` substring to locate automatically the correct class for a graph, so graphs created with a version can be used the the other; of course, graphs with more than `Integer.MAX_VALUE` nodes cannot be loaded with the standard version.

In the case of WebGraph, we expect a longer coexistence. New developments will probably happen in the standard version, and will be ported as soon as possible to the big version.