Distributed programming using POP-Java

Beat Wolf, University of applied science Fribourg, University of Würzburg
beat.wolf@hefr.ch

Abstract

The POP-Model is an innovative way to distribute objects over the network, while at the same time simplifying parallel programming. A first implementation of this model, called POP-C++, has been realized as an extension of the C++ programming language. An implementation as an extension of the Java programming language, called POP-Java, started as a bachelor project. This first prototype of POP-Java basically reimplemented the functionality of POP-C++. We adapted this prototype to better integrate with the Java language and fix some remaining problems. The POP-Java implementation was benchmarked against RMI, both implementing the same algorithm. POP-Java yielded comparable results in terms of performance. Now that the foundation of POP-Java has been put in place, expanding it to allow the usage of clouds and run on the windows operating system is a priority.

Keywords: Java; Distributed; Parallel, OO programming;

1 Introduction

Despite progresses made in programming distributed environments this activity remains complex. Indeed often complex code is needed to distribute an algorithm over the network and complex thread synchronization needs to be done to scale over multiple cores and over multiples computers. These are the problems the POP-Model tries to solve thanks to the introduction of the notion of parallel class. Several new keywords have been defined allowing the programmer to easily distribute objects and define how concurrent calls to methods are handled. POP-C++ [1] was the first language to implement these notions and these keywords, allowing for a completely transparent distribution of the objects. The only thing that the programmer needs to do is to annotate the classes he wants to distribute with the appropriate keywords defining how the methods of distributed objects are called. The programmer can declare methods as synchronous or asynchronous, and restrict their concurrent access using 3 keywords: concurrent, sequential and mutex. The details on how those keywords work can be found in [1] and [2].

2 Work Accomplished

A prototype of POP-Java implementing most of the basic features was developed by Valentin Clément [2]. This prototype uses a similar approach than POP-C++ by adding new keywords to the java language. Thus, a new programming language was created, called POP-Java. Because of the added keywords, the POP-Java syntax was no longer compatible with the standard Java language. To correct this, we defined a new version of the POP-Java language using Java annotations instead of keywords, resulting in a syntax that is completely compatible with the standard Java language. As in the original POP-Java prototype, there is still the need for a parser that translates POP-Java source code into java, but only to include some boiler plate code to make programming easier and not to change the syntax.

3 Benchmark

To validate the POP-Java in terms of functionality and performance, the DNA sequence aligner presented last year [3], was ported to use POP-Java. This allowed for a direct comparison between the RMI and POP-Java performance of the same algorithm. The algorithm also consist of many short calls on remote objects, where as the POP-Model was designed to be used for fewer but longer calls. This usecase was an opportunity to optimize POP-Java under those circumstances, increasing performance overall. The test was done on a local grid installation, using 1-6 machines. Those machines where equipped with quad-core CPUs and connected trough a 1GB/s network. The performance test was done using a dataset of 4 billion sequences of an average length of 120 bases, aligned against the human chromosome 7. The first machine only uses 3 cores of the 4 available to align the data, this is to keep one core available for the read and write operations, making sure that they don't bottleneck the algorithm. For this reason, the optimal speedup is adapted. Figure 1 shows the speedup over 6 machines.

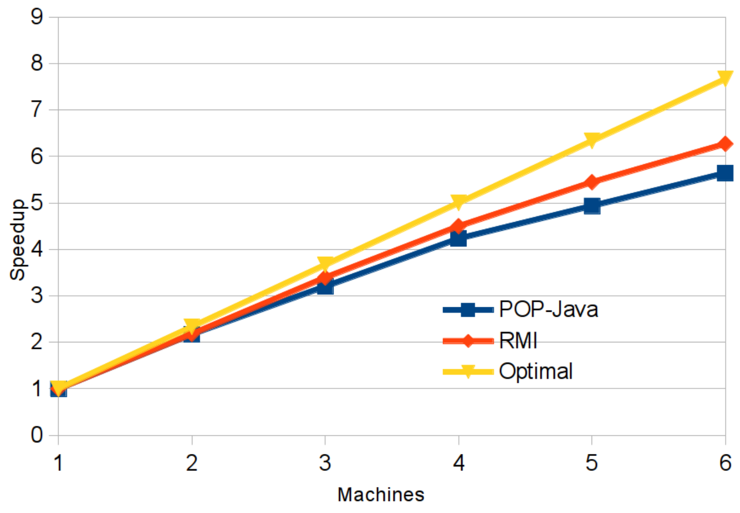

Figure 1: Speedup over 6 machines, with the first machine only using 3 cores

We can see that the speedup progression of both implementations is similar. For the absolute difference in execution speed, RMI is 13% faster on one machine and 30% on 6 machines than the POP-Java version. The exact cause of this difference has yet to be found.

4 Work in Progress and Future Work

While POP-Java works well under linux, much is still left to do. Now that distributing objects over multiple computers in a grid like environment works well, the next step is to bring POP-Java to the cloud. Currently an exploratory project is evaluating this possibility. One of the goals is to make the distribution on a cloud just as easy as the distribution on a grid, possibly by adding a new annotation similar to *@POPConfig(Type.URL)*. Additionally to be able to distribute the application over more network configurations, windows will also be targeted. With java beeing a multiplatform language, windows support is the natural next step after linux and osx. The main obstacle to port POP-Java to windows is the way how objects are instanciated on remote machines. Currently POP-Java opens a secure SSH connection to the remote machine and executes the command needed to start the required object. In a windows environment, setting up SSH is too complicated for most users, specially with the premise of POP-Java to make distributed programming easier. How this will be solved is still an open question that we hope to address soon. Further improvements are also planned to make the usage of POP-Java easier. The currently used intermediate step between .pjava and pure java files complicates the compilation of application using POP-Java. Making this step either easier or remove it completely is a goal of further development

References

[1] T. A. Nguyen, P. Kuonen, “Programming the Grid with POP-C++,“ in *Future Generation Computer Systems (FGCS)*, N.H. Elsevier, Volume 23, Issue 1, 1 January 2007, pages 23-30.

[2] Valentin Clément, Pierre Kuonen, “POP-Java, Bachelor thesis,“ at *University of applied sciences, Fribourg*, 2010

[3] Beat Wolf, Pierre Kuonen, David Atlan, “General purpose distributed DNA aligner,“at the *Doctoral Workshop on Distributed Systems*, 2012 available at `http://beat.wolf.home.hefr.ch/documents/aligner-vuedesalpes.pdf`