The APGAS Library: Resilient Parallel and Distributed Programming in Java 8

Olivier Tardieu
IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
tardieu@us.ibm.com

Abstract

We propose the APGAS library for Java 8. Inspired by the core constructs and semantics of the Resilient X10 programming language, APGAS brings many benefits of the X10 programming model to the Java programmer as a pure, idiomatic Java library.

APGAS supports the development of resilient distributed applications running on elastic clusters of JVMs. It provides asynchronous lightweight tasks (local and remote), resilient distributed termination detection, and global heap references.

We compare and contrast the X10 and APGAS programming styles, review design choices, and demonstrate that APGAS achieves performance comparable with X10.

Categories and Subject Descriptors D.1.3 [Programming Techniques]: Concurrent Programming—distributed programming

Keywords APGAS, Java, PGAS, X10

1. Overview

The APGAS programming model is Asynchronous Partitioned Global Address Space—is a simple but powerful model of concurrency and distribution. It combines PGAS with asynchrony. In an APGAS the computation and data in an application are logically partitioned into places. In APGAS the computation is further organized into lightweight asynchronous tasks following an async-finish structure. Concretely a place is an abstraction of a mutable, shared-memory region and worker threads operating on this memory, typically realized as an operating system process. Memory locations in one place can contain global references to locations at other places. An application starts with a main task. Tasks can spawn local and remote asynchronous tasks (async capability). A task can wait for the completion of all the tasks transitively spawned from it (finish capability).

The X10 programming language is an imperative, object-oriented language built upon the APGAS model. Recently X10 has been enriched to support failure-aware and elastic programming with the design and implementation of Resilient X10. Resilient X10 applications can detect the loss of a place—the data and computation at this place—and implement recovery strategies. Since X10 v2.5.1, they can also request and make use of new places when running on dynamic execution platforms, e.g., in the cloud.

In this work we propose to realize the Resilient APGAS programming model not as a language such as X10 but as an API, i.e., a library, for a mainstream language: Java 8. APGAS is open source and available at [http://x10-lang.org](http://x10-lang.org). Our contributions are:

- We implement Resilient APGAS as a library for Java 8.
- We compare X10 with APGAS in Java 8.
- We implement the Unbalanced Tree Search benchmark using APGAS and compare performance with Java and X10.

2. Programming with APGAS

Figures 1 and 2 compare X10's HelloWorld program with its implementation in Java 8 with APGAS. This program spawns a task at each place to print a message on the console.

Syntax. The code in Figure 1 uses four X10 constructs: finish, at, async, and here. In contrast the code in Figure 2 uses imported static methods of the APGAS Constructs class such as method "static void asyncAt(Place p, SerializableJob job)". In our experience programmers find X10's "at(place) async" idiom confusing so we replace it with a single invocation.

```java
class HelloWorld {
    public static void main(String[] args) {
        finish(() -> {
            for (final Place place : places()) {
                asyncAt(place, () -> {
                    System.out.println("Hi from " + here);
                });
            };
        });
    }
}
```

Figure 1. HelloWorld in X10.

```java
import static apgas.Constructs.*;
import apgas.Place;

class HelloWorld {
    public static def main(Rail[String]) {
        finish {
            for (place in Place.places) {
                at(place) async {
                    Console.OUT.println("Hi from " + here);
                }
            }
        }
    }
}
```

Figure 2. HelloWorld in Java 8 with APGAS.
Block statements in X10 become lambdas in APGAS. In particular, `SerializableJob` is a functional interface. The resulting mix of `"(" and ")"`; is by far the biggest annoyance with APGAS.

**Serialization.** All X10 objects are serializable and the X10 compiler implements serialization and deserialization methods for all X10 classes. In contrast APGAS relies on Java for serialization and serialized classes have to extend `java.io.Serializable`.

In HelloWorld for instance, the innermost lambda is serialized to the destination place. Thanks to type inference, this lambda is inferred to be of type `SerializableJob`, which extends `java.io.Serializable`.

To alleviate runtime serialization exceptions, we provide an optional Eclipse plugin for APGAS that generates compiler warnings when serializable lambdas capture non-serializable objects.

**Boxing.** X10 permits accessing local variables and initializing local values from inner tasks.

```java
var p:int=3n; val q:int; finish async q=p; val r=q;
```

In Java, we have to box these variables and values.

```java
final int p[] = new int[1]; p[0] = 3;
final int q[] = new int[1];
finish((() -> async(() -> q[0] = p[0]));
final int r = q[0];
```

The X10 to Java compiler inserts such boxes so the performance is equivalent. But before that, the X10 compiler can verify that p and q are properly initialized before use and that q is never mutated after initialization. APGAS has no such capabilities.

3. **Design and Implementation**

X10 is compiled to C++ or Java. Its design reflects this duality. For instance, X10 supports structures in addition to classes. Generic types in X10 resemble C++ templates. Of course APGAS adopts Java idioms throughout. Moreover the APGAS implementation exploits services of the JVM and Java libraries whenever possible, e.g., the fork/join framework, Java serialization, and Java collections.

APGAS is built on top of the Hazelcast in-memory data grid [3]. Like APGAS, Hazelcast is an open source framework implemented in Java and deployed as a jar file. APGAS relies on Hazelcast to (i) connect and coordinate elastic, distributed clusters of JVMs, (ii) invoke remote tasks via its distributed executor service, and (iii) protect critical runtime and application data from failures.

The APGAS library implements the core elements of the APGAS programming model: lightweight tasks, distributed termination detection, and global heap references. Exceptions escaping from tasks are collected by the innermost enclosing finish. By setting the `apgas.resilient` system property, the application can request resilient versions of these core elements. Remote task invocations fail gracefully when the destination place is unavailable. Resilient finish ensures `happen-before` invariance [2].

APGAS supports elasticity. Places can be added to a running application by simply launching a new JVM with the `ip:port` address of an existing JVM in the cluster. For convenience, we implement two alternative launchers to start multiple places at once either on the localhost or, using Hadoop YARN, in a distributed system. Applications can register a callback that is invoked when a place is added or has failed.

The APGAS library is currently implemented in about 2,000 non-blank, non-comment lines of Java code. About a third of this code implements distributed termination detection.

1 Data generated using David A. Wheeler’s SLOCCount.