Collective Relocation for Associative Distributed Collections of Objects

Daisuke Fujishima, Graduate School of System Informatics, Kobe University, Kobe, Japan
Tomio Kamada, Advanced Institute for Computational Science, RIKEN, Wako, Japan

ABSTRACT

The field of parallel computing has experienced an increase in the number of computing nodes, allowing broader applications, including computations that have irregular features. Some parallel programming languages handle object data structures and offer marshaling/unmarshaling mechanisms to transpose them. To manage data elements across computing nodes, some research on distributed collections has been conducted. This study proposes a distributed collection library that can handle multiple collections of object elements and change their distributions while maintaining associativity between their elements. This library is implemented on an object-oriented parallel programming language, X10. The authors assume pairs of associative collections such as vehicles and streets in a traffic simulation. When many vehicles are concentrated on streets assigned to certain computing nodes, some of these streets should be moved to other nodes. The authors’ library assists the programmer in easily distributing the associative collections over the computing nodes and collectively relocating elements while maintaining the data sharing relationship among associative elements. The programmer can describe the associativity between objects by using both declarative and procedural methods. They show a preliminary performance evaluation of their library on a Linux cluster and the K computer.

KEYWORDS
Collective Relocation, Concurrent Aggregates, Distributed Collections Serialize, X10

1. INTRODUCTION

The field of parallel computing has experienced an increase in the number of computing nodes. In large-scale parallel computers, each computing node has its own local memory space, and programmers determine allocation of data and tasks to the computing nodes. Parallel computing has widened its application to include computations that have irregular features. In these applications, allocation of data and tasks not only becomes more important but also more difficult. Some parallel programming languages handle object data structures and offer marshaling/unmarshaling mechanisms to transpose them. To manage data elements spread over computing nodes, some mechanisms for distributed aggregates are proposed (Chien & Dally, 1990; The Charm++ Parallel Programming System Manual, 2016). Charm++ also offers a data redistribution mechanism for dynamic load balancing.

This study proposes a distributed collection library that can change the distribution of multiple object collections that have associativity between their elements. The library is implemented for X10 (Saraswat, Bloom, Peshansky, Tardieu & Grove, 2015), which is a parallel programming language that uses a partitioned global address space (PGAS) memory model. X10 allows object data structures and offers distributed arrays (called DistArray) and a global load balancing library. We assume the...
relocation of two or more associative distributed collections such as an array of particles and their
calculation or vehicles and streets in a traffic simulation.

In the case of a traffic simulation, streets and vehicles are represented as objects. To avoid
communication overhead, vehicles on a street should be allocated to a computing node to which the
street is assigned. When a vehicle moves to a street that is assigned to a different computing node,
the vehicle must be transferred to the corresponding computing node. If vehicles concentrate on
streets that are assigned to a computing node, the programmer will want to redistribute streets and
associative vehicles for dynamic load balancing.

Our library supports the collective relocation of object collections. It utilizes object serialization
of X10, and the reference relationship among objects is evenly maintained between the associative
elements of different distributed collections. Programmers can specify a new distribution of associative
collections by using both declarative and procedural methods in our application programming
interface (API). Object relocation is synchronously executed by all the computing nodes, and the
library uses collective communications among nodes to reduce communication overhead. To manage
the associativity between multiple distributed collections, our library offers a type of a key-value
mechanism. The programmer can use the key-value relationship to specify the distributions of
associative collections. The programmer can change the key-value relationship based on the states
of elements during the object relocation or independently with the relocation.

Section 2 summarizes the memory model of X10 and introduces its distributed collection library.
Section 3 introduces some case studies in which programmers want to relocate associative aggregates
while maintaining the data sharing relationship among their elements. Section 4 explains the API
design of our library, and Section 5 briefly describes the current implementation of this library. Section
6 shows a preliminary evaluation of this library on a Linux cluster and the K computer. Section 7
reviews related studies, and Section 8 provides the conclusions of the study.

2. MEMORY MODEL OF X10

X10 is a parallel programming language developed by IBM Research and adopts the PGAS model.
Figure 1 illustrates the memory model of X10. The address space is partitioned using places, in which
each object belongs to a specific place. Each place represents a shared-memory multiprocessor. An
activity represents a sequential computation. Each place can run one or more activities.

X10 allows activities to change places. For example, at (place) S for statement S creates an
activity at the place and executes S synchronously. To execute S, the values to which S can refer to are
recursively copied to the place. Serializer of X10 recognizes the graph structure of reachable objects,
and Deserializer reproduces the structure at the destination. X10 features asynchronous programs,
and also allows fork-join-style programs. async S begins a new activity that executes S in the current
place, and at (place) async S asynchronously creates an activity that executes S on the place. finish S
awaits the termination of all activities spawned by S including those remotely or indirectly created.

X10 also offers cross-place references. GlobalRef class allows us to create a global reference to
a local object. The global reference can be passed to other places; however, activities can access the
body object only when they are running at the place in which the body object resides.

X10 offers certain types of distributed objects. PlaceLocalHandle can represent a set of objects
that are allocated at different places. When activities are spawned to the places, the activities can
access the “branch” object at each place by using PlaceLocalHandle. DistArray[T] represents a
distributed array spread over places, where T is a type parameter to specify the content type. DistArray
can accept any type as its content type, including primitive types and Object classes. We can specify
the distribution of a DistArray at its creation, and this distribution remains unchanged after creation.
Each element is indexed by Point in a k-dimensional space for k≥1. a(p) accesses the element of a
distributed array a for a point p when the target element is mapped to the place at which the activity
is running. If the distribution does not map the given point to the current place, an exception is raised.
Using Executable Slicing to Improve Rogue Software Detection Algorithms
Jan Durand, Juan Flores, Travis Atkison, Nicholas Kraft and Randy Smith (2011). 
*International Journal of Secure Software Engineering* (pp. 53-64). 
[www.igi-global.com/article/using-executable-slicing-improve-rogue/55269?camid=4v1a](www.igi-global.com/article/using-executable-slicing-improve-rogue/55269?camid=4v1a)

On IT-Modelling in a Cross-Competence World
[www.igi-global.com/chapter/modelling-cross-competence-world/23423?camid=4v1a](www.igi-global.com/chapter/modelling-cross-competence-world/23423?camid=4v1a)