Resilient and Elastic APGAS

Olivier Tardieu
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA

This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0002, by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under Award Number DE-SC0008923, and by the Air Force Office of Scientific Research under Contract No. FA8750-13-C-0052.
Motivation

- **Problem**
  - faults are increasingly common in distributed systems
  - available resources vary dynamically
  - sensibility to faults and recovery strategies vary

- **Our approach**
  - contain faults within a shared-memory domain (process)
  - support programming fault tolerance and resource management
  - bake in only fundamental capabilities; build the rest as libraries
  - focus on in-memory fault-tolerance

<table>
<thead>
<tr>
<th>Don’t care</th>
<th>Programmable</th>
<th>Transparent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional MPI, X10</td>
<td>ULFM MPI, Resilient X10/APGAS</td>
<td>Hadoop, Spark, C/R</td>
</tr>
<tr>
<td>fast, not fault tolerant</td>
<td>flexible</td>
<td>slow, ad hoc</td>
</tr>
</tbody>
</table>
Motivation

- Application-specific failure recovery
  - redo or approximate
  - shrink or compensate resource loss
  - exploit semantics of computation
    - conservation of mass, incompressibility...
  - exploit structure of computation

- Example: KMeans clustering
  - algorithm: SPMD, iterative refinement
  - resilient algorithm: synchronous checkpoints

- Example: Unbalanced Tree Search
  - algorithm: distributed work-stealing scheduler
  - resilient algorithm: asynchronous checkpoints
Outline

- APGAS programming model
  - principles
  - APGAS in X10
  - toy examples
    - X10, Java, Scala

- Resilience and elasticity
  - principles
    - failure model
    - resilient control
    - resilient data
  - implementation
  - applications
  - performance
  - formalization
APGAS
The APGAS Programming Model

- Asynchronous PGAS = PGAS + Tasks

- PGAS: Partitioned Global Address Space
  - computation and data is partitioned across places
  - global heap but explicit remote references

- Tasks
  - asynchronous tasks for *everything*
    - parallelism and distribution, regular and irregular
    - motivation: heterogeneous systems, irregular workloads
  - distributed termination detection

- Implementations
  - X10 language
  - library for Java and Scala
APGAS Constructs in X10

Task creation
- `async s`
- `at(p) async s`

Distributed termination detection
- `finish s`

Places
- `here`
- `Place.places()`

Global heap
- `GlobalRef[T]`
APGAS Idioms in X10

- Task parallelism
  ```scala
  async runThis(); runThat();
  ```

- Remote task
  ```scala
  at(p) async runThere(arg);
  ```

- Divide-and-conquer parallelism
  ```scala
  def fib(n:Long):Long {
    if(n < 2) return n;
    val f1:Long; val f2:Long;
    finish {
      async f1 = fib(n-1);
      f2 = fib(n-2);
    }
    return f1 + f2;
  }
  ```

- SPMD
  ```scala
  finish for(p in Place.places()) {
    at(p) async runEverywhere();
  }
  ```

- Remote evaluation – the explicit way
  ```scala
  val ref = GlobalRef(new Cell(0));
  finish {
    at(p) async {
      val tmp = evalThere(arg);
      at(ref) async ref().set(tmp);
    }
  }
  val v = ref().get();
  ```

- Remote evaluation – the better way
  ```scala
  val v = at(p) evalThere(arg);
  ```
public class BlockDistRail[T] {  
    protected val sz:Long; // block size  
    protected val raw:PlaceLocalHandle[Rail[T]]; // storage  

    public def this(sz:Long, places:Long){T haszero} {  
        this.sz = sz;  
        raw = PlaceLocalHandle.make[Rail[T]](PlaceGroup.make(places), ()=>new Rail[T](sz));  
    }  

    public operator this(i:Long) = (v:T) { at(Place(i/sz)) raw()(i%sz) = v; }  
    public operator this(i:Long) = at(Place(i/sz)) raw()(i%sz);  
}  

public class Foo {  
    public static def main(Rail[String]) {  
        val rail = new BlockDistRail[Long](5, 4);  
        rail(7) = 8; Console.OUT.println(rail(7));  
    }  
}
HelloWorld in X10 and Java APGAS

- **X10**

```java
finish for (place in Place.places()) {
  at(place) async Console.OUT.println("Hello from " + here);
}
```

- **Java APGAS**

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

finish(() -> {
  for (final Place place : places()) {
    asyncAt(place, () -> System.out.println("Hello from " + here()));
  }
});
```

imported static methods

Java 8 no-arg lambdas
def fib (n: Int) : Int = {
  if(n < 2) n else {
    var v0,v1 = 0
    finish {
      async {
        v0 = fib (n - 2)
      }
      v1 = fib (n - 1)
    }
    v0 + v1
  }
}
Resilience and Elasticity
Outline

- APGAS programming model
  - principles
  - APGAS in X10
  - toy examples
    - X10, Java, Scala

- Resilience and elasticity
  - principles
    - failure model
    - resilient control
    - resilient data
  - implementation
  - applications
  - performance
  - formalization
Principles

- Elasticity and failure model
  - places can be added, removed, or spontaneously fail
    - fail-stop model: instantaneous loss of tasks and heap at failed place
    - failures reported with exceptions, callbacks, polling

- Resilient control
  - task at failed place are lost
  - execution continues at healthy places
    - including orphaned tasks
    - execution order is preserved: happens-before invariance

- Resilient data
  - data at failed place is lost
  - data in resilient stores is preserved unless failure is catastrophic
    - resilient stores do not belong to any place (but shards can be hosted within places)
    - resilient stores are designed to transparently overcome non-catastrophic failures
Dead Place Exceptions

- Use exceptions to report place failure
  - synchronously on task spawn
  - asynchronously during task execution

```java
try {
    finish {
        try {
            at(Place(1)) async runThere();
        } catch (e:DeadPlaceException) {
            Console.ERR.println("DeadPlaceException caught when spawing task");
        }
    }
} catch (e:MultipleException) {
    if (e.getExceptionsOfType[DeadPlaceException]().size > 0) {
        Console.ERR.println("DeadPlaceException caught while running task");
    }
}
```
Resilient Control

- No deadlocks due to place failures
  - no waiting on lost tasks

- No data races due to place failures
  - no loss of ordering dependencies

```java
// main task
val g = new GlobalVariable(0);
try {
    at(Place(1)) { // task A
        at(Place(0)) { // task B
            g() = 42;
        } // task B
    } // task A
} catch (e:DeadPlaceException) {}
g() = 0;
```
Resilient Stores

- In-memory distributed concurrent key/value maps
  - atomic operations
    - single key: insertion, retrieval, update, removal
    - sets of keys: snapshot...
  - accessible from any place
  - copy semantics

- Manage data replication and/or data movement
  - configurable fault tolerance

- Compliant or relaxed semantics
  - delayed writes from failed places?
  - two usage models
    - keep compliant store after failure
    - clone relaxed store after failure
Implementation
Implementations

- Resilient elastic transport
  - TCP/IP + SSH
  - TCP/IP + Hadoop YARN
  - ULFM MPI (not elastic yet)
  - Hazelcast

- Resilient store
  - X10 implementations
    - place zero stores
    - double in-memory store
  - Hazelcast distributed map

- Resilient control
  - place zero finish
  - Hazelcast finish
Applications and Performance
Case Studies

- Teams
  - resilient barrier, all-reduce...

- Unbalanced Tree Search

- LULESH

- Main-Memory Map-Reduce engine (M3R)
  - KMeans, Hidden Markov Model trainer

- Executor for iterative SPMD applications

- Global Matrix Library
  - linear regression, logistic regression, pagerank
Unbalanced Tree Search

- Count nodes in a randomly generated tree

- Lifeline-based global distributed load balancing
  - no central scheduler: mix of work stealing and dealing
  - no synchronization other than distributed termination detection

- Uncoordinated incremental checkpoints
  - each worker periodically checkpoints state (nodes processed so far, remaining)
  - source worker checkpoints source and destination state when transferring work

- Coordinated recovery
  - abort all workers, load and shrink checkpoint, restart all workers

- Performance evaluation
  - running 22 places with 8 workers each on 22 nodes with 8 cores each
Unbalanced Tree Search

Million nodes traversed per second per place

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>COMPLIANT PLACE 0</th>
<th>RELAXED PLACE 0</th>
<th>HAZELCAST STORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential</td>
<td>100%</td>
<td>94%</td>
<td>96%</td>
<td>95%</td>
</tr>
<tr>
<td>distributed</td>
<td>98%</td>
<td>91%</td>
<td>92%</td>
<td>83%</td>
</tr>
<tr>
<td>no failure</td>
<td>6.51</td>
<td>6.12</td>
<td>6.22</td>
<td>6.16</td>
</tr>
<tr>
<td>1 failure</td>
<td>6.35</td>
<td>5.89</td>
<td>6.01</td>
<td>5.40</td>
</tr>
<tr>
<td>3 failures</td>
<td>6.00</td>
<td>5.60</td>
<td>5.63</td>
<td>4.30</td>
</tr>
</tbody>
</table>

23
KMeans

- **KMeans on Resilient M3R**
  - 1,000,000 of 3D points were categorized into 8 clusters

![Graph showing execution time of each iteration for Resilient X10 and Normal (non-elastic) X10.]

- In the Resilient X10 execution, 5 places were killed here.
- In the Resilient X10 execution, 10 places were added here.
- Resilient X10 (10 places)
- Normal (non-elastic) X10 (10 places)
- Running on 5 places
- Running on 15 places

4.5 sec to reconfigure Resilient X10 runtime, and reload data from R.S.
Formalization
# Semantics of (Resilient) X10

**Values**
\[
\begin{align*}
v & ::= o \mid o^p \mid E \mid \text{DPE} \\
\end{align*}
\]

**Expressions**
\[
\begin{align*}
e & ::= v \mid x \mid e.f \mid \{f:e, \ldots, f:e\} \mid \text{globalref}\ e \mid \text{valof}\ e \\
\end{align*}
\]

**Statements**
\[
\begin{align*}
s & ::= \text{skip;} \mid \text{throw}\ v \mid \text{val}\ x = e\ s \mid e.f = e; \mid \{s\ t\} \\
& \quad \text{at}(p)\text{val}\ x = e\ \text{in}\ s \mid \text{async}\ s \mid \text{finish}\ s \mid \text{try}\ s\ \text{catch}\ t \\
& \quad \text{at}(p)\ s \mid \text{async}\ s \mid \text{finish}_{\mu}\ s \\
\end{align*}
\]

**Configurations**
\[
\begin{align*}
k & ::= \langle s, g \rangle \mid g \\
\end{align*}
\]

**Global heap**
\[
g ::= \emptyset \mid g \cdot [p \mapsto h] \\
\]

**Local heap**
\[
h ::= \emptyset \mid h \cdot [o \mapsto (\tilde{f}_i : \tilde{v}_i)]
\]

---

**TX10**

- **object id**
- **global object id**
- **exception**

---

**no atomic, no clocks**

---

Semantics of (Resilient) X10 [ECOOP’14] by Crafa, Cunningham, Saraswat, Shinnar, Tardieu
Conclusion

- Tremendous challenge!
  - correctness and performance
- Tremendous opportunity!
- Many open research questions

Try APGAS!
http://x10-lang.org