

On the Design and Implementation of the Parsimony Checkpoint Package

Bruno R. Preiss
Electrical and Computer Engineering
University of Waterloo

<http://www.pads.uwaterloo.ca/Bruno.Preiss/talks/1998/padsgroup3/slides.ps>

What is checkpointing and why do I need it?

- PADS—parallel and distributed *simulation*
- *time warp* or *optimistic* simulation
- avoid *deadlock*
- method: *speculative* computation
- *speculative parallelism* breaks the speed-of-light barrier!

So what's the problem?

- speculative computation is sometimes wrong
- speculative computation changes the system state
- sometimes need to *rollback* to a previous state
- therefore need to *save* (checkpoint) a previous states
- **application programmer should not have to worry about this**

How did Yaddes do it?

- required user to statically declare all system state variables
- provided user with a *reference* (pointer) to current state
- (standard CS solution to all problems—add one more level of indirection)
- user must explicitly dereference state pointer to update system state
- rollback to earlier state by changing the pointer

The problem with the Yaddes way of doing it

- user had to declare state statically—dynamic data structures are hard to do
- user code had to dereference pointers—yuck
- cannot use any library code or canned software that was not aware of the need to checkpoint
- all the great new language features (inheritance, polymorphism) are inaccessible

Wish List

- eliminate need for static declaration of checkpointable data
- checkpoint dynamically allocated, arbitrarily linked data structures
- checkpoint the state of instances of user-defined classes
- **make checkpointing transparent**
- write classes without need to know they may be checkpointed

The Genesis of the Parsimony Checkpoint Package

- Java supports *object serialization* for object persistence and for remote method invocation
- realized that this facility could be used to construct a *checkpoint* of an object
- serialization saves an object and the transitive closure comprised of every object accessible from the initial object!

How to you checkpoint an object

- Java provides a `ObjectStream` classes that support reading/writing objects from/to a byte stream
- to checkpoint an object you write it into a byte array!
- to recover a checkpointed object you read it from a byte array!

Snapshot Class

```
public class Snapshot
{
    protected long time;
    protected byte[] buffer;

    public Snapshot (long time, Checkpointable object) throws ...
    {
        try
        {
            this.time = time;
            ByteArrayOutputStream byteStream = new ByteArrayOutputStream ();
            ObjectOutputStream objectStream =
                new ObjectOutputStream (byteStream);
            objectStream.writeObject (object);
            buffer = byteStream.toByteArray ();
        }
        catch (IOException e) { ... }
    }
}
```

```
public long getTime ()
    { return time; }

public Checkpointable getObject () throws ...
{
    try
    {
        ByteArrayInputStream byteStream = new ByteArrayInputStream (buffer);
        ObjectInputStream objectStream = new ObjectInputStream (byteStream);
        return (Checkpointable) objectStream.readObject ();
    }
    catch (IOException e) { ... }
    catch (ClassNotFoundException e) { ... }
}
}
```

Adding that Level of Indirection

- How do you change the “state” of an object without affecting the user of that object?
- define the *interface* for a checkpointable object
- have users interact with a “proxy” object that implements the same interface

What does the Proxy Do?

- the proxy forwards all methods calls to methods in the interface to the object instance that is the current state
- the proxy makes snapshots of the current state when requested to do so
- the proxy rolls back to a previously checkpointed state when requested to do so

Do I need to write a proxy?

- implemented a *proxy compiler*
- to be checkpointable, all the interfaces *implemented* by a class must extend (directly or indirectly) the `Parsimony.Checkpoint.Checkpointable` interface:

```
package Parsimony.Checkpoint;
```

```
public interface Checkpointable extends Serializable  
{  
}
```

An Example—A Checkpointable Interface

```
package Parsimony.Checkpoint.Demos;  
  
public interface SampleInterface  
    extends Parsimony.Checkpoint.Checkpointable  
{  
    void setValue (int value);  
}
```

An Example—A Checkpointable Class

```
package Parsimony.Checkpoint.Demos;

public class SampleClass implements SampleInterface
{
    protected Integer value;

    public void setValue (int value)
        { this.value = new Integer (value); }

    public String toString ()
        { return value.toString (); }
}
```

Checkpoint Proxy Class—Produced by the Compiler

```
// Checkpoint proxy created by Parsimony.Checkpoint.Compiler  
// DO NOT EDIT.
```

```
package Parsimony.Checkpoint.Demos;
```

```
public class SampleClass_Proxy  
    extends Parsimony.Checkpoint.CheckpointableObject  
    implements Parsimony.Checkpoint.Demos.SampleInterface  
{  
    public void setValue (int arg0)  
    {
```

```
((SampleClass) getRef()).setValue(arg0);
```

```
}
```

```
}
```

Methods provided by the CheckpointableObject class?

`createProxy(Checkpointable obj)` creates a proxy for the given object (static method)

`checkpoint(long time)` checkpoints this object

`rollback(long time)` rolls this object back

`commit(long time)` fossil collects all states older than time

Using the checkpointable object

```
public class Demo
{
    public static void main (String[] args)
    {
        try
        {
            SampleInterface obj = (SampleInterface)
                CheckpointableObject.createProxy (new SampleClass());

            obj.setValue (57);
            System.out.println (obj);
            CheckpointableObject.checkpoint (obj, 1);

            obj.setValue (1234);
            System.out.println (obj);
            CheckpointableObject.checkpoint (obj, 2);

            obj.setValue (654321);
            System.out.println (obj);
            CheckpointableObject.checkpoint (obj, 3);
        }
    }
}
```

```
        CheckpointableObject.rollback (obj, 2);
        System.out.println (obj);

        CheckpointableObject.rollback (obj, 1);
        System.out.println (obj);
    }
    catch (Exception e) { ... }
}
}
```

So how do they do that?

- checkpoint compiler is implemented in Java
- it does not read or parse Java source code
- instead it **dynamically loads** the class for which a proxy is required and then uses Java's *reflection* support to get all the information it needs!!!
- compiler then writes out the Java source code for the proxy class and invokes `javac` to compile it!!!

Future Work

- need to port to Java 1.2 which provides a `MarshaledObject` class
- need to assess the performance of all this
- need to investigate *incremental* state saving (deltas)
- wild idea: can you implement “sparse” checkpointing by saving a list of the methods invoked and their arguments between checkpoint operations?

Incremental Checkpointing

- exploit “rollback locality”
 - probability of rolling back to a given state decreases with the age of that state
- save most recent state whole
- save sequence of *deltas* to apply in order to back up

Computing Deltas

- based on work of Myers (1986)
- copy/insert algorithm
 - copy values from current state to new state
 - insert values in new state from delta

Modified Algorithm

- copy/insert/replace algorithm
 - copy values from current state to new state
 - insert values in new state from delta
 - replace values in new state from delta

Preliminary Results

setting	compression ratio	time ratio
maximum	24.5	1.48
normal	18.7	1.47
fast	16.8	1.39