



# A Resilient Framework for Iterative Linear Algebra Applications in X10

Sara S. Hamouda

Australian National University

Josh Milthorpe

IBM T.J. Watson Research Center

Peter E. Strazdins

Australian National University

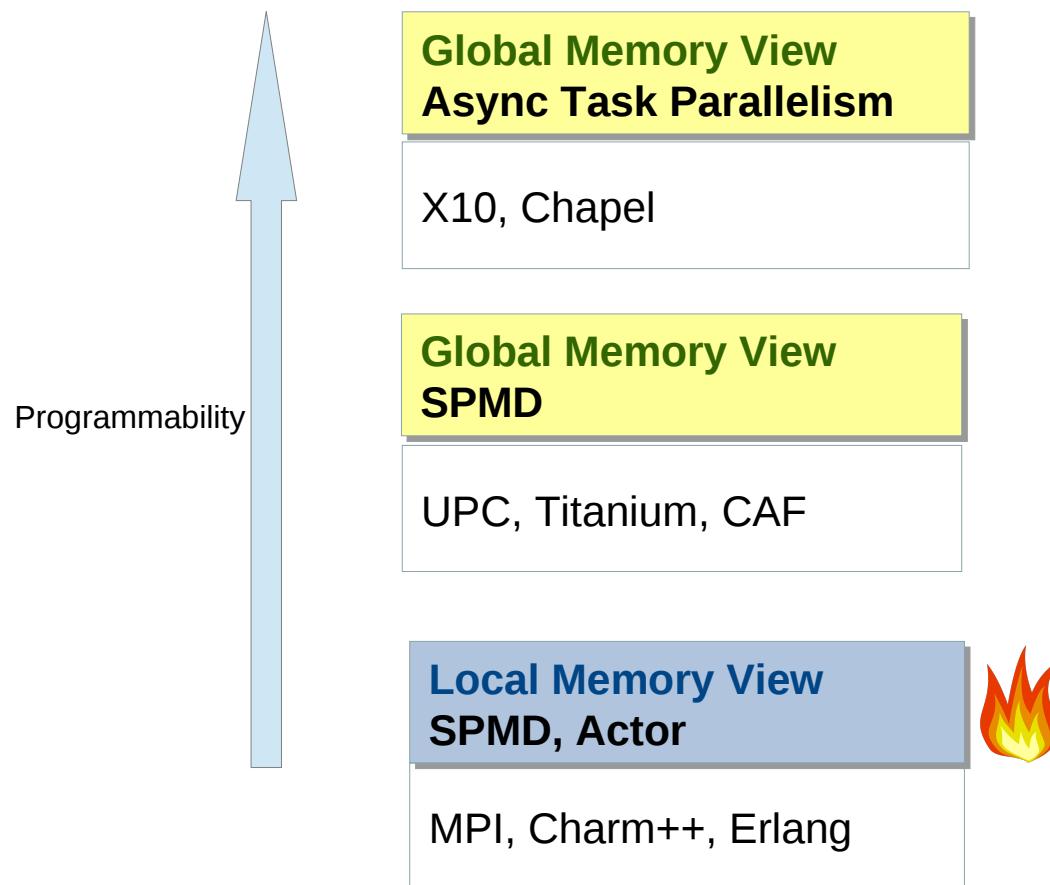
Vijay Saraswat

IBM T.J. Watson Research Center

**2015 ACM SIGPLAN  
X10 Workshop at PLDI**



# Programmability vs. Resilience





# PPoPP 2014 - Resilient X10 Paper

## Resilient X10

Efficient failure-aware programming

David Cunningham<sup>2</sup> \*, David Grove<sup>1</sup>, Benjamin Herta<sup>1</sup>, Arun Iyengar<sup>1</sup>, Kiyokuni Kawachiya<sup>3</sup>, Hiroki Murata<sup>3</sup>, Vijay Saraswat<sup>1</sup>, Mikio Takeuchi<sup>3</sup>, Olivier Tardieu<sup>1</sup>

<sup>1</sup>IBM T. J. Watson Research Center

<sup>2</sup>Google Inc.

<sup>3</sup>IBM Research - Tokyo

dcunnin@google.com, {groved,bherta,aruni,vsaraswa,tardieu}@us.ibm.com, {kawatiya,mrthrk,mtake}@jp.ibm.com

# X10 Domain Specific Libraries

- GML (Global Matrix Library)
- ANUChem
- ScaleGraph
- M3RLite (Main Memory Map Reduce Lite)
- Megaffic (Traffic flow simulation)
- SatX10 (Parallel boolean satisfiability)



# X10 Domain Specific Libraries

- **Resilient GML** (Global Matrix Library)
- ANUChem
- ScaleGraph
- M3RLite (Main Memory Map Reduce Lite)
- Megaffic (Traffic flow simulation)
- SatX10 (Parallel boolean satisfiability)



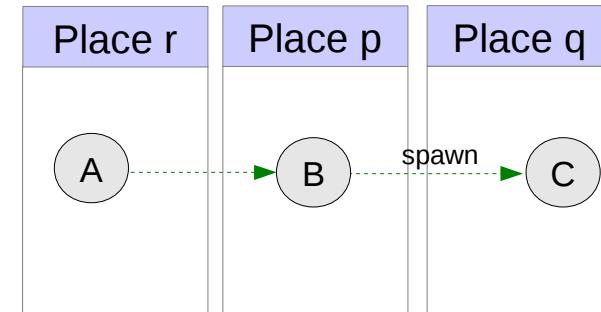
# Outline

- Resilient X10
- GML
  - API Overview
  - Resilience Limitations
  - Resilience Enhancements
  - Performance Results



# Resilient X10

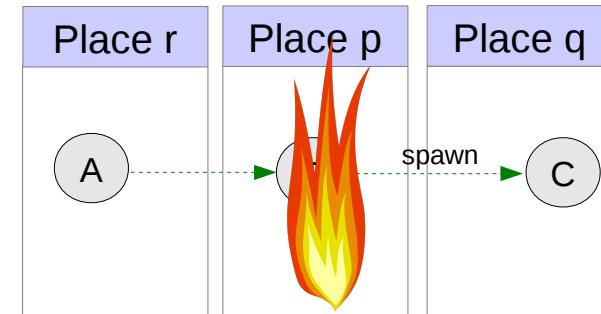
```
// Task A
try {
    at(p) {
        // Task B
        finish {
            at(q) async {
                // Task C
            }
        }
    }
} catch (dpe:DeadPlaceException) {
    // recovery step
}
// D
```





# Resilient X10

```
// Task A
try {
    at(p) {
        // Task B
        finish {
            at(q) async {
                // Task C
            }
        }
    }
} catch (dpe:DeadPlaceException) {
    // recovery step
}
// D
```





# Resilient X10

```
// Task A
try {
    at(p) {
        Place.r
        Place.p
        Place.a
    }
} catch (dpe:DeadPlaceException) {
    // recovery step
}
// D
```

- Resilient X10 supports only the **sockets** backend
- Resilient Store
  - Centralized Store
  - Distributed Store (currently not supported)



# Outline

- Resilient X10
- GML
  - API Overview
  - Resilience Limitations
  - Resilience Enhancements
  - Performance Results



# Global Matrix Library (GML)

- Distributed matrix library in X10
- Simple programming model
  - Matrix based
  - Sequential style programming
  - Efficient iterative processing
- Potential compilation target for high-level array languages
  - Provides fundamental vector/matrix routines
  - Supports dense and sparse matrix formats
  - Uses BLAS and LAPACK



# GML Vector/Matrix Classes

Single Place	Multi-Place		
	Duplicated	Distributed	
		1 Block/Place	N Blocks/Place
DenseMatrix SymDense TriDense SparseCSC SparseCSR	DupDenseMatrix DupSparseMatrix	DistDenseMatrix DistSparseMatrix	DistBlockMatrix
Vector	DupVector	DistVector	



# PageRank Implementation in GML

```
/* Matrix dimensions */  
var m:Long, n:Long;  
/* Matrix partitioning configurations */  
var rowBlocks:Long, colBlocks:Long, rowPlaces:Long,  
colPlaces:Long;  
/* Create GML objects */  
val G:DistBlockMatrix = DistBlockMatrix.make(m, n, rowBlocks,  
colBlocks, rowPlaces, colPlaces);  
val P:DupVector = DupVector.make(n);  
val U:DistVector = DistVector.make(n, G.getAggRowBs());  
val GP:DistVector = DistVector.make(n, G.getAggRowBs());  
  
/* Data initialization code omitted */
```

**Algorithm:**  
for (1..k)

$$P = \alpha G P + (1 - \alpha) E U^T P$$

# PageRank Implementation in GML

```
/* Data initialization code omitted */

for (1..k) {
    GP.mult(G, P).scale(alpha);
    val UtP1a = U.dot(P) * (1-alpha);
    GP.copyTo(P.local());
    P.local().cellAdd(UtP1a);
    P.sync();
}
```

## Algorithm:

```
for (1..k)
```

$$P = \alpha G P + (1 - \alpha) U^T P$$



# Outline

- Resilient X10
- GML
  - API Overview
  - Resilience Limitations
  - Resilience Enhancements
  - Performance Results



# GML Resilience Limitations

- Fixed place distribution
- Failure of a place resulted in loss of GML objects
  - no built-in mechanism for restoring objects



# Resilience Enhancements (1)

- Arbitrary and dynamic place distribution
  - `make(..., places:PlaceGroup)`
  - `remake(..., newPlaces:PlaceGroup)`



# DistVector Redistribution

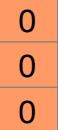
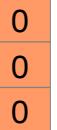
```
val pg = make_P0_P2_group();
```

```
A.remake(pg);
```

Before remake

Place 0	Place 1	Place 2
 		

After remake

Place 0	Place 1	Place 2
 		



# Resilience Enhancements (2)

- Added in-memory snapshot / restore capability to GML classes

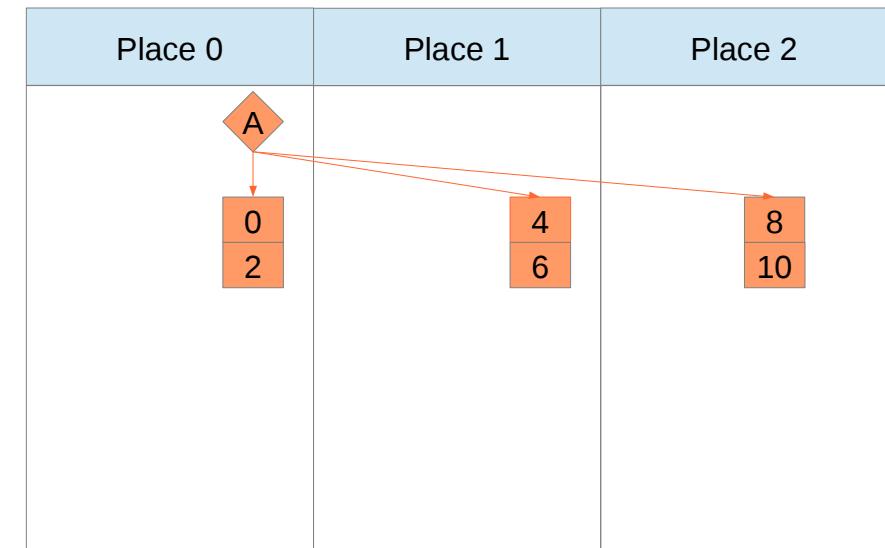
```
interface Snapshottable {  
    makeSnapshot():Snapshot;  
    restoreSnapshot(Snapshot):void;  
}
```



# DistVector Snapshot/Restore

◇ A PlaceLocalHandle

```
val A = DistVector.make(6);
A.init((i:Long)=> i*2.0);
```



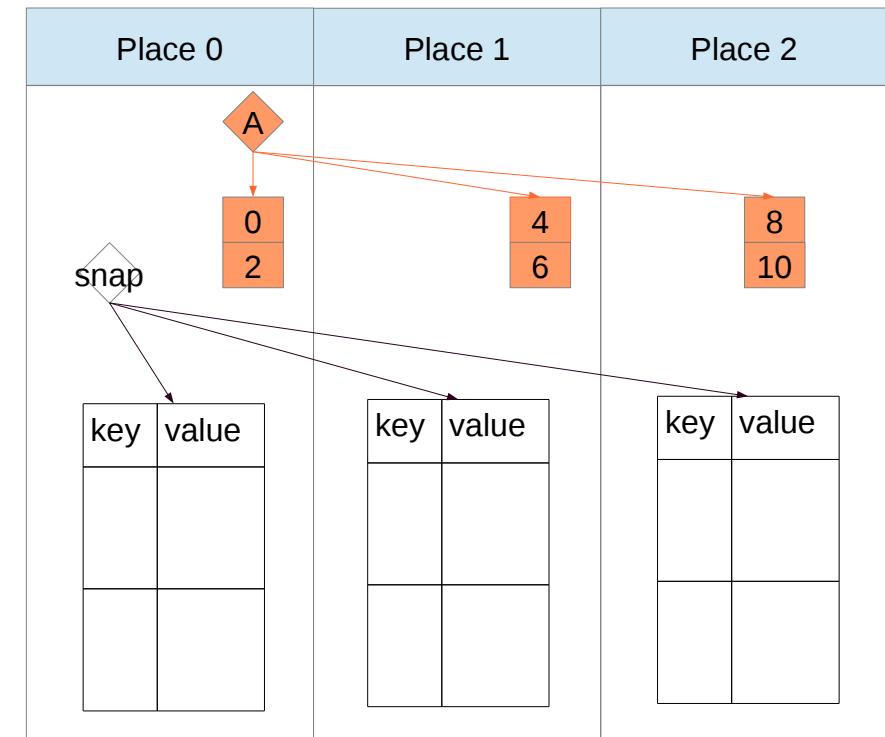


# DistVector Snapshot/Restore

◇ A PlaceLocalHandle

```
val A = DistVector.make(6);
A.init((i:Long)=> i*2.0);
```

```
val snap = A.makeSnapshot();
```



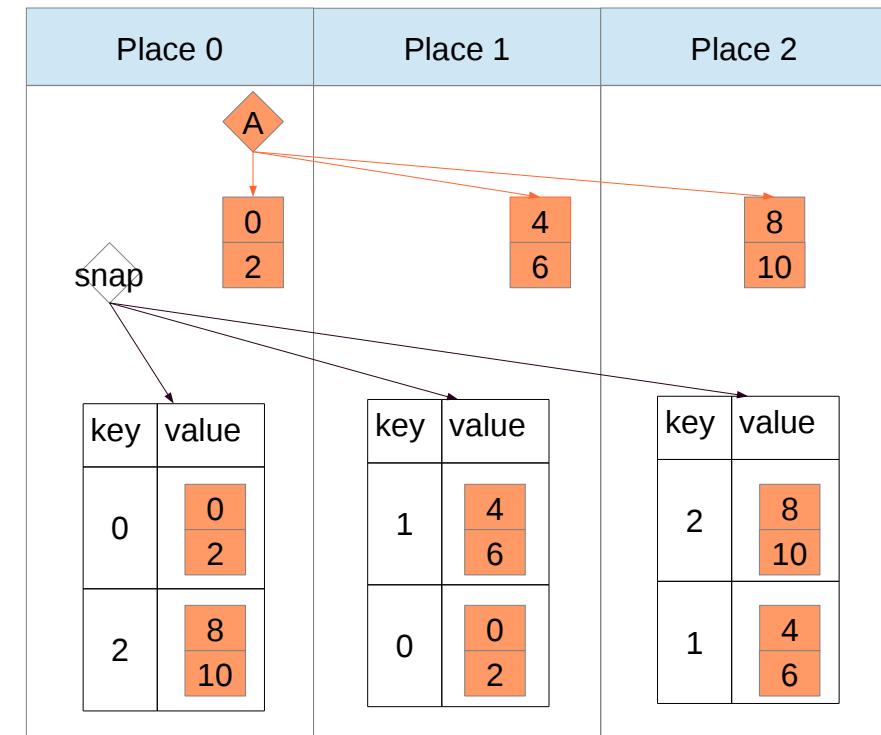


# DistVector Snapshot/Restore

```
val A = DistVector.make(6);  
A.init((i:Long)=> i*2.0);
```

```
val snap = A.makeSnapshot();
```

◆ A PlaceLocalHandle  
→ Copy from Snapshot





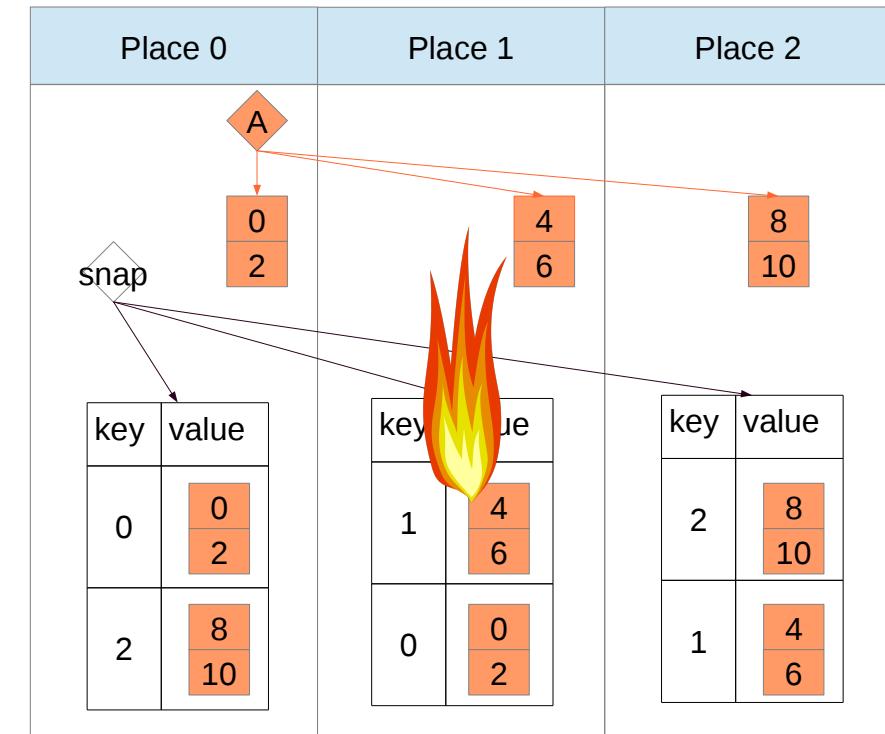
# DistVector Snapshot/Restore

◇ A PlaceLocalHandle

```
val A = DistVector.make(6);
A.init((i:Long)=> i*2.0);
```

```
val snap = A.makeSnapshot();
```

*/\* Place 1 failed \*/*





# DistVector Snapshot/Restore

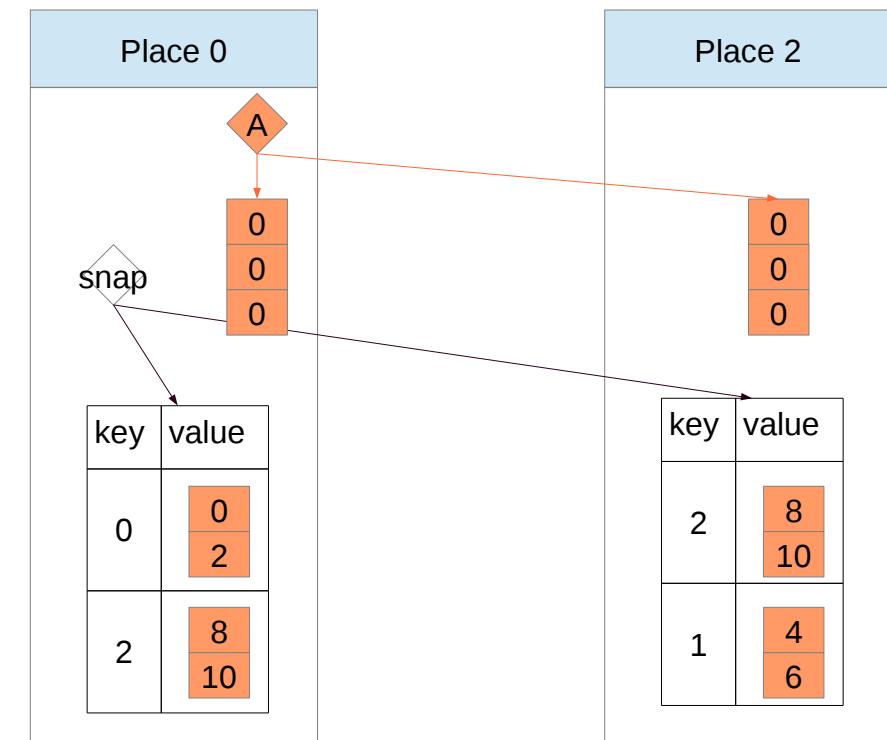
◇ A PlaceLocalHandle

```
val A = DistVector.make(6);
A.init((i:Long)=> i*2.0);
```

```
val snap = A.makeSnapshot();
```

*/\* Place 1 failed \*/*

```
val pg = make_P0_P2_group();
A.remake(pg);
```





# DistVector Snapshot/Restore

◇ A PlaceLocalHandle

→ Copy from Snapshot

```
val A = DistVector.make(6);  
A.init((i:Long)=> i*2.0);
```

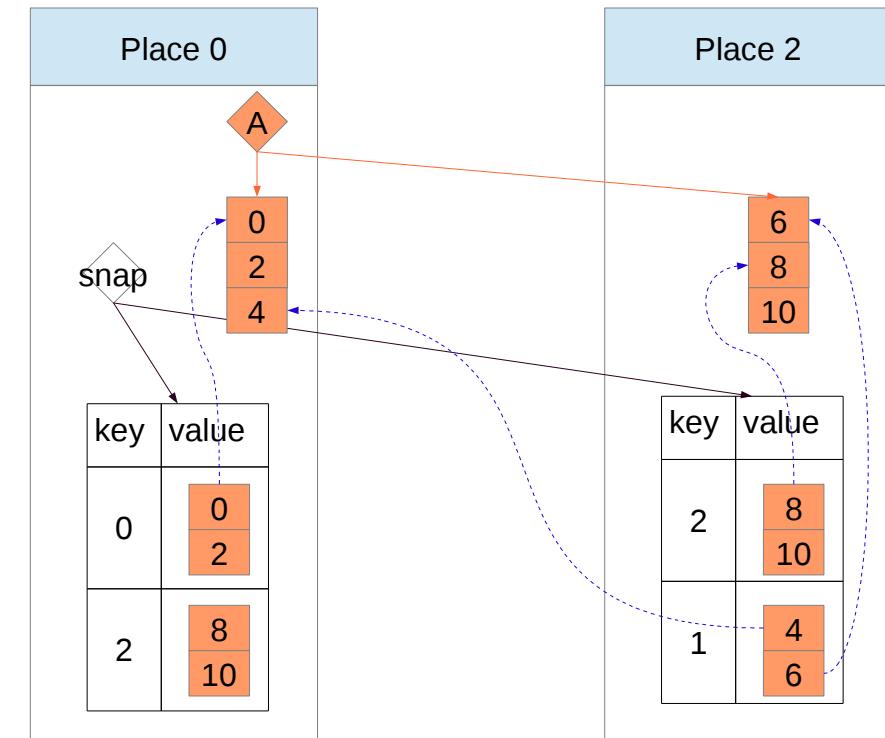
```
val snap = A.makeSnapshot();
```

*/\* Place 1 failed \*/*

```
val pg = make_P0_P2_group();
```

```
A.remake(pg);
```

```
A.restoreSnapshot(snap);
```





# (1) Iterative Programming Model

```
interface ResilientIterativeApp {  
    def step():void;  
    def isFinished():void;  
    def checkpoint(store:AppResilientStore):void;  
    def restore(newPlaces:PlaceGroup,  
              store:AppResilientStore,  
              snapshotIter:Long):void;  
}
```



## (2) Iterative Application Executor

```
val store:AppResilientStore;
while (!isFinished()) {
    try {
        if (restoreRequired) {
            val newPlaces = createRestorePlaceGroup();
            restore(newPlaces, store, checkpointIter);
        }
        step();
        if (iter % checkpointInterval == 0) {
            checkpoint(store);
            checkpointIter = iter;
        }
        iter++;
    } catch (dpe:DeadPlaceException) {
        restoreRequired = true;
    }
}
```



## (3) Application Resilient Store

- Concurrent and atomic snapshot/restore for multiple GML objects

```
class AppResilientStore {  
    def startNewSnapshot();  
    def save(obj:Snapshottable);  
    def saveReadOnly(obj:Snapshottable);  
    def commit();  
    def cancelSnapshot();  
    def restore();  
}
```



# PageRank Snapshot/Restore

```
def checkpoint(store:AppResilientStore){  
    store.startNewSnapshot();  
    store.saveReadOnly(G);  
    store.saveReadOnly(U);  
    store.save(P);  
    store.commit();  
}
```

```
def restore(newPlaces:PlaceGroup,  
          store:AppResilientStore, snapshotIter:Long){  
    G.remake(..., newPG);  
    U.remake(..., newPG);  
    P.remake(newPG);  
    store.restore();  
    //restore other primitive variables  
}
```

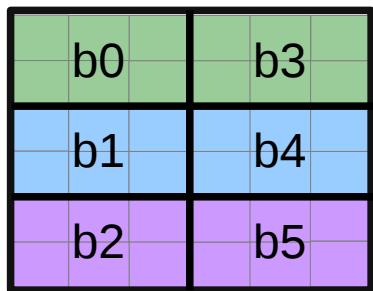


## (4) Restore Modes

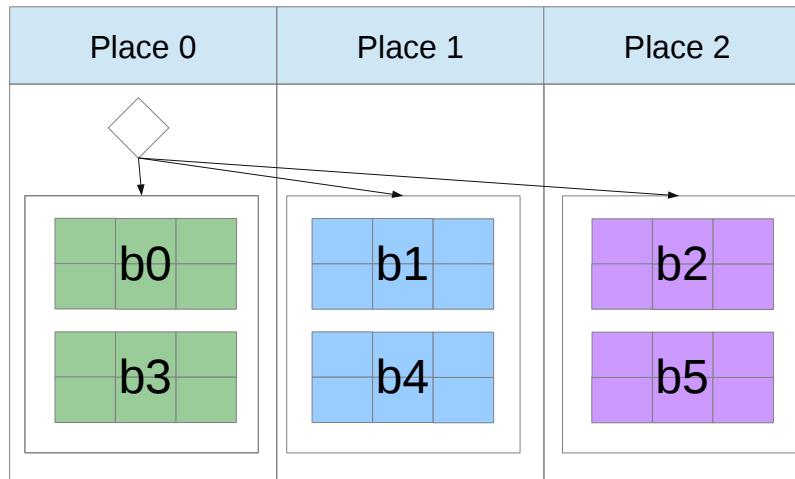
- Restoration Modes
  - **Shrink**
  - **Shrink-Rebalance**
  - **Replace Redundant**



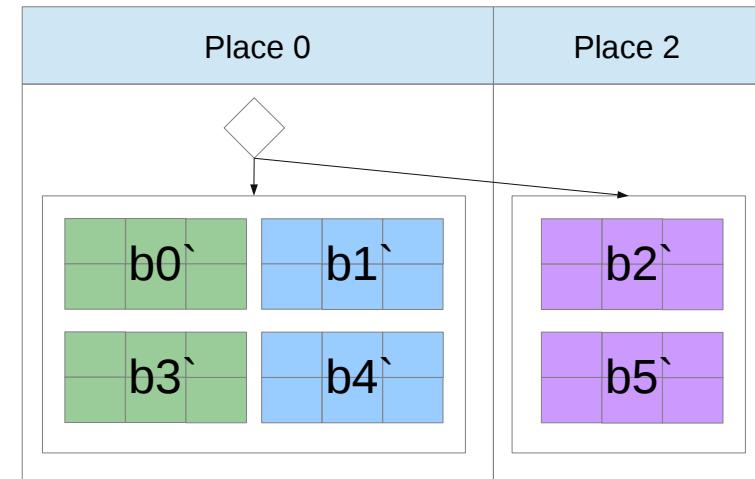
# Shrink



Before remake

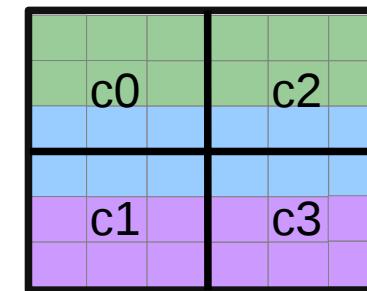
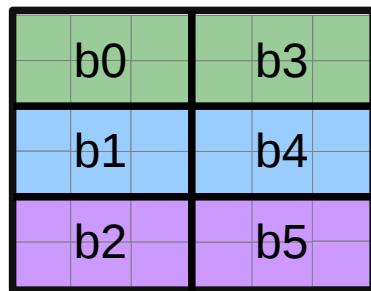


After remake

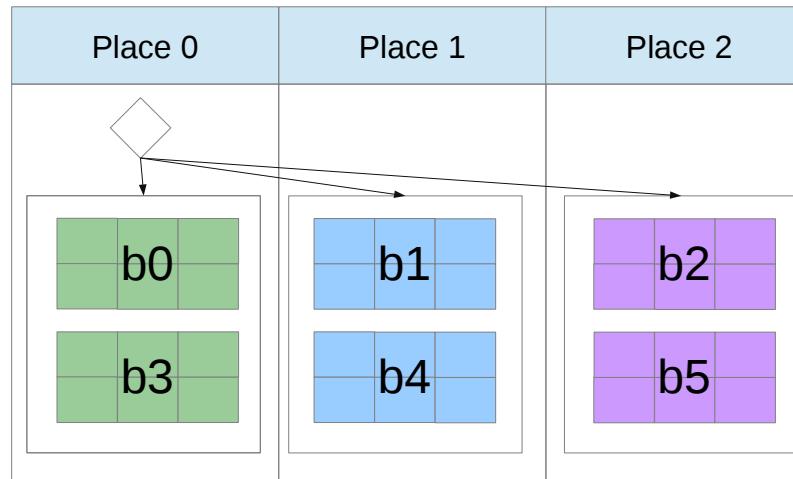




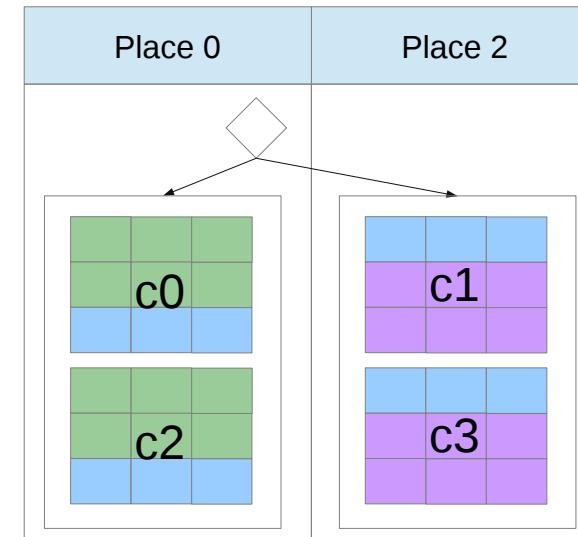
# Shrink Rebalance



Before remake



After remake



# Outline

- Resilient X10
- GML
  - API Overview
  - Resilience Limitations
  - Resilience Enhancements
  - Performance Results



# Experimental Setup

- SoftLayer Cluster host hosted at IBM Almaden Research Center
  - 11 nodes: four-core 2.6 GHz Intel Xeon E5-2650 CPU with 8 GB of memory
- X10:
  - Native X10, version 2.5.2
  - 4 places per node, `X10_NTHREADS=1`
  - X10RT sockets backend
- GML:
  - OpenBLAS version 0.2.13  
(`OPENBLAS_NUM_THREADS=1`)



# Checkpoint and Restore Overheads

- Checkpoint every 10 iterations (3 checkpoints per run)
- A single place failure at iteration 15
- Repeat the experiments with different restore modes:
  - Shrink
  - Shrink-Rebalance
  - Redundant

# Applications

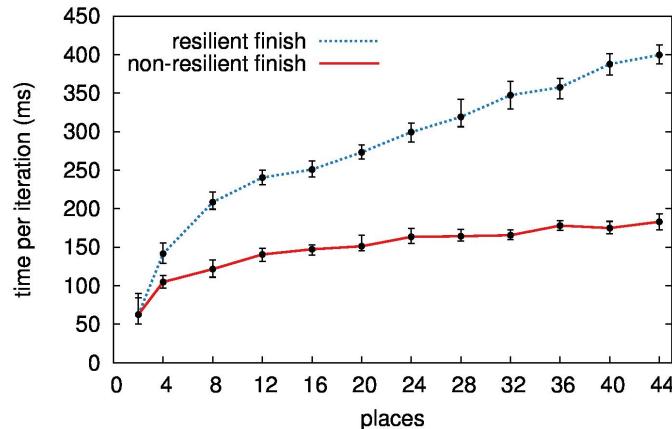
- Dense
  - LinReg (50,000 X 500 per place)
  - LogReg (50,000 X 500 per place)
- Sparse
  - PageRank (2M edges per place)



# Resilient X10 Overhead

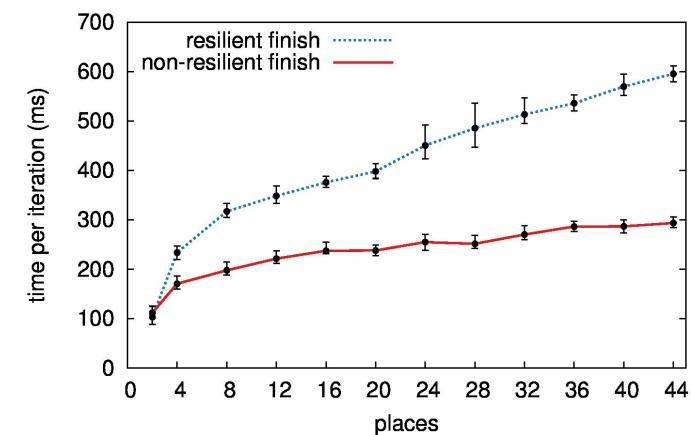
## Linear Regression

Overhead on 44 places: ~120%



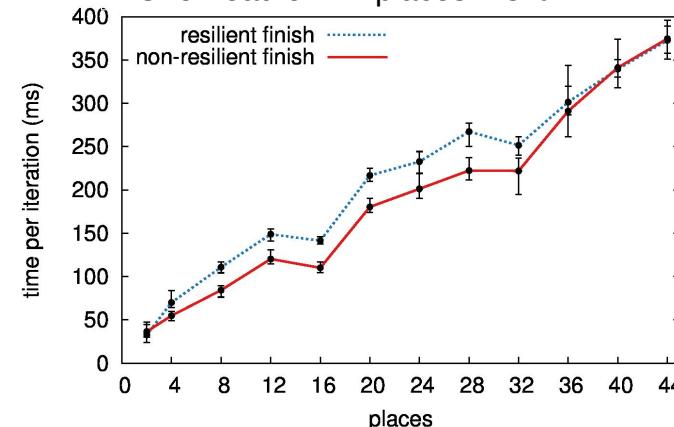
## Logistic Regression

Overhead on 44 places: ~100%



## PageRank

Overhead on 44 places: ~3%





# Checkpoint and Restore Overheads

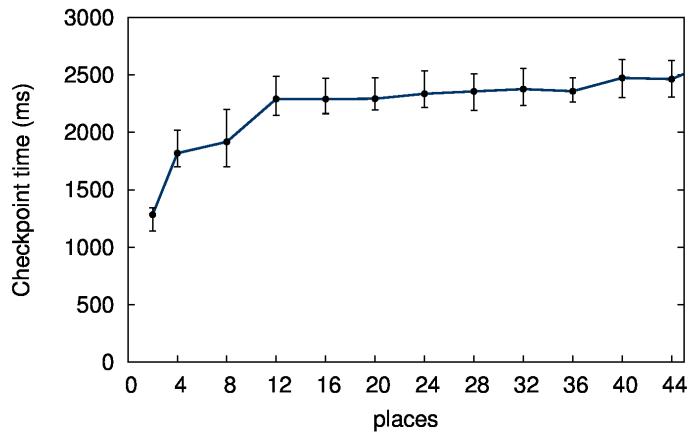
- Checkpoint every 10 iterations (3 checkpoints per run)
- A single place failure at iteration 15
- Repeat the experiments with different restore modes:
  - Shrink
  - Shrink-Rebalance
  - Redundant



# Time per Checkpoint

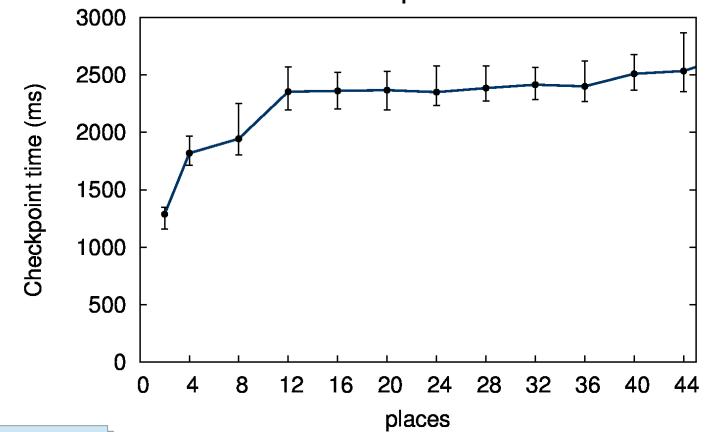
## Linear Regression

Overhead from 12 to 44 places: ~8%



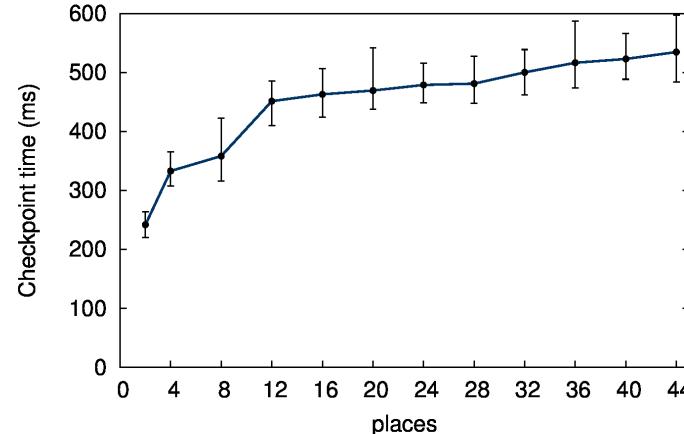
## Logistic Regression

Overhead on 44 places: ~8%



## PageRank

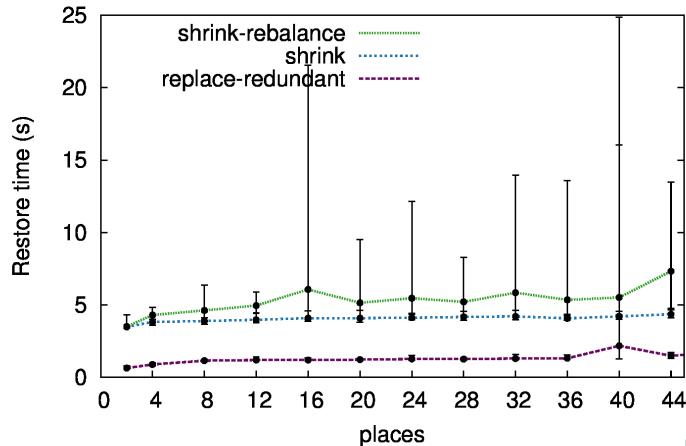
Overhead from 12 to 44 places: ~18%





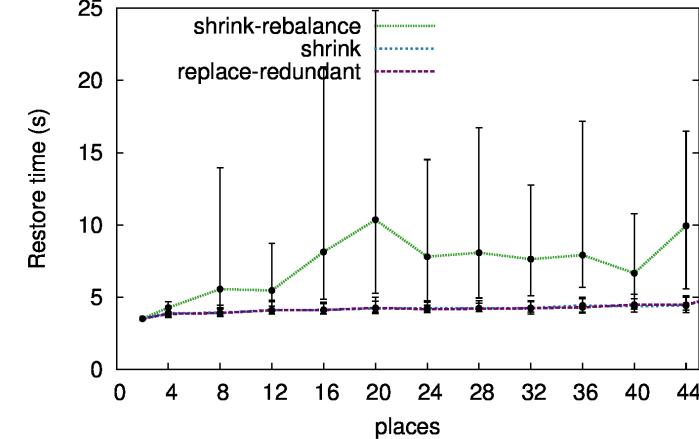
## Linear Regression

From 12 to 44 places in Redundant mode: ~25%



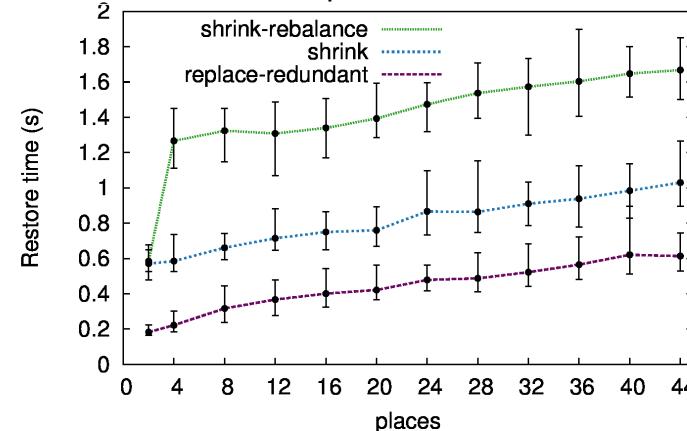
## Logistic Regression

From 12 to 44 places in Redundant mode: ~9%



## PageRank

From 12 to 44 places in Redundant mode: ~67%

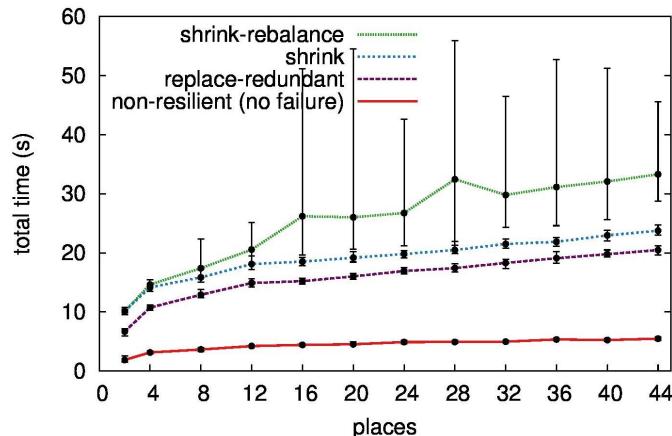




# 3 Checkpoints + 1 Restore

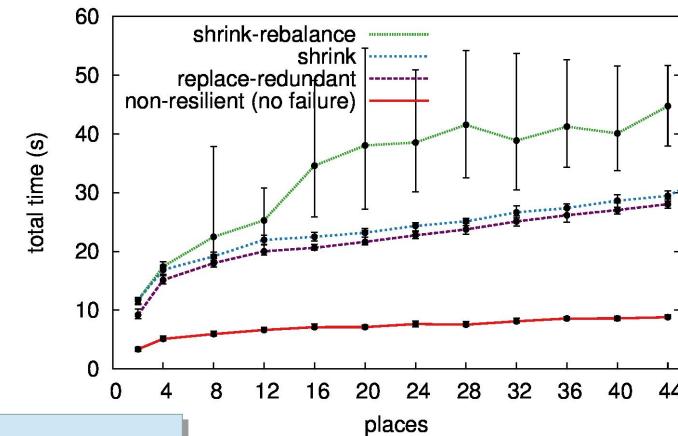
## Linear Regression

Overhead to Non-Resilient X10 mode: ~273%



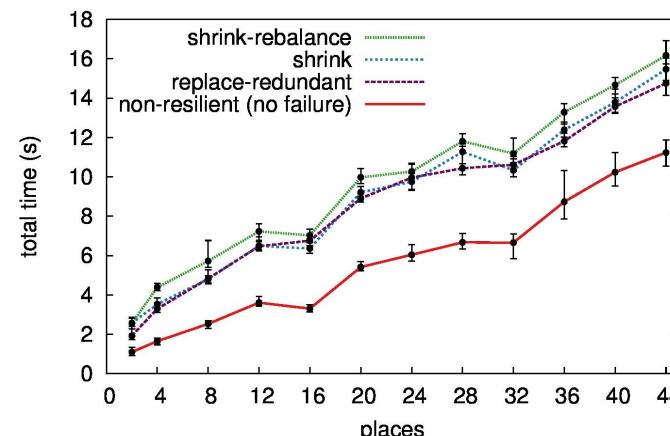
## Logistic Regression

Overhead to Non-Resilient X10 mode: ~219%



## PageRank

Overhead to Non-Resilient X10 mode: ~31%





# Conclusions

- We presented a framework for developing resilient linear algebra applications using X10
  - Simple to use
  - Generic enough to be used in other libraries (i.e. ScaleGraph)
  - Assumptions:
    - Place 0 is immortal
    - Fails when 2 neighbouring processes fail
  - Reasonable scalability for dense matrix checkpoint / restore
  - *Main source of the performance overhead is due to Resilient X10 mode itself*
- Full source code freely available at <http://x10-lang.org> as part of GML version 2.5.2



# Future Work

- Improve Resilient GML's performance:
  - Enhancements in Resilient X10
    - Support MPI, avoid the centralized resilient store
  - More efficient fault tolerance techniques
  - Use Elastic X10
- Compare Resilient GML with other frameworks (i.e. Spark).