Supporting Fault Tolerance in a Data-Intensive Computing Middleware

Tekin Bicer, Wei Jiang and Gagan Agrawal
Department of Computer Science and Engineering
The Ohio State University

IPDPS 2010, Atlanta, Georgia
Motivation

- Data Intensive computing
  - Distributed Large Datasets
  - Distributed Computing Resources
  - Cloud Environments
- Long execution time
- High Probability of Failures
A Data Intensive Computing API

FREERIDE

```
{ * Outer Sequential Loop * }
While() {
    { * Reduction Loop * }
    Foreach(element e) {
        (i, val) = Process(e);
        RObj(i) = Reduce(RObj(i), val);
    }

    Global Reduction to Combine RObj
}
```

**Reduction Object** represents the intermediate state of the execution

- Reduce func is commutative and associative
- Sorting, grouping, overheads are eliminated with red. func/obj.
Simple Example

3 5 8 4 1 3 5 2 6 7 9 4 2 4 8

Our large Dataset

Our Compute Nodes

Local Reduction (+)

Robj[1] = 28

Local Reduction (+)

Robj[1] = 25

Local Reduction (+)

Robj[1] = 24

Result = 71

Global Reduction (+)

IPDPS, 2010
Remote Data Analysis

- Co-locating resources gives best performance…
- But may not be always possible
  - Cost, availability etc.
- Data hosts and compute hosts are separated
- Fits grid/cloud computing
- FREERIDE-G is a version of FREERIDE that supports remote data analysis
Fault Tolerance Systems

- **Checkpoint based**
  - System or Application level snapshot
  - Architecture dependent
  - High overhead

- **Replication based**
  - Service or Application
  - Resource Allocation
  - Low overhead
Outline

- Motivation and Introduction
- Fault Tolerance System Approach
- Implementation of the System
- Experimental Evaluation
- Related Work
- Conclusion
A Fault Tolerance System based on Reduction Object

- Reduction object...
  - represents intermediate state of the computation
  - is small in size
  - is independent from machine architecture
- Reduction obj/func show associative and commutative properties

Suitable for Checkpoint based Fault Tolerance System
An Illustration

Local Reduction (+)

Robj = 0

Robj = 0

Robj = 21

Robj = 21
Modified Processing Structure for FTS

```c
{ * Initialize FTS * }
While {
  Foreach ( element e ) {
    (i, val) = Process(e);
    RObj(i) = Reduce(RObj(i), val);
    { * Store Red. Obj. * }
  }
  if ( CheckFailure() )
  { * Redistribute Data * }
  { * Global Reduction * }
}
```
Outline

- Motivation and Introduction
- Fault Tolerance System Design
- Implementation of the System
- Experimental Evaluation
- Related Work
- Conclusion
Simple Implementation of the Alg.

- Reduction object is stored another comp. node
  - Pair-wise reduction object exchange
- Failure detection is done by alive peer
Demonstration

Redistribute Failed Node’s Remaining Data

Failure Detected

Final Result
Outline

- Motivation and Introduction
- Fault Tolerance System Design
- Implementation of the System
- Experimental Evaluation
- Related Work
- Conclusion
Goals for the Experiments

- Observing reduction object size
- Evaluate the overhead of the FTS
- Studying the slowdown in case of one node’s failure
- Comparison with Hadoop (Map-Reduce imp.)
Experimental Setup

- **FREERIDE-G**
  - Data hosts and compute nodes are separated

- **Applications**
  - K-means and PCA

- **Hadoop (Map-Reduce Imp.)**
  - Data is replicated among all nodes
Experiments (K-means)

- Without Failure Configurations
  - Without FTS
  - With FTS
- With Failure Configuration
  - Failure after processing %50 of data (on one node)

Execution Times with K-means 25.6 GB Dataset

- Reduction obj. size: 2KB
- With FT overheads: 0 - 1.74%
  - Max: 8 Comp. Nodes, 25.6 GB
- Relative: 5.38 – 21.98%
  - Max: 4 Comp. Nodes, 25.6 GB
- Absolute: 0 – 4.78%
  - Max: 8 Comp. Nodes, 25.6 GB
Experiments (PCA)

- Reduction obj. size: 128KB
- With FT overheads: 0 – 15.36%
  - Max: 4 Comp. Nodes, 4 GB
- Relative: 7.77 – 32.48%
  - Max: 4 Comp. Nodes, 4 GB
- Absolute: 0.86 – 14.08%
  - Max: 4 Comp. Nodes, 4 GB

Execution Times with PCA, 17 GB Dataset
Comparison with Hadoop

- **w/f = with failure**
- Failure happens after processing 50% of the data on one node

<table>
<thead>
<tr>
<th></th>
<th>Hadoop</th>
<th>Hadoop w/f</th>
<th>FREERIDE-G</th>
<th>FREERIDE-G w/f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overheads</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hadoop</td>
<td>23.06</td>
<td>71.78</td>
<td>78.11</td>
<td></td>
</tr>
<tr>
<td>FREERIDE-G</td>
<td>20.37</td>
<td>8.18</td>
<td>9.18</td>
<td></td>
</tr>
</tbody>
</table>

K-means Clustering, 6.4GB Dataset
Comparison with Hadoop

- One of the comp. nodes failed after processing 25, 50 and 75% of its data

Overheads
- Hadoop
  32.85 | 71.21 | 109.45
- FREERIDE-G
  9.52 | 8.18 | 8.14

K-means Clustering, 6.4GB Dataset, 8 Comp. Nodes
Outline

● Motivation and Introduction
● Fault Tolerance System Design
● Implementation of the System
● Experimental Evaluation
● Related Work
● Conclusion
Related Work

- Application level checkpointing
  - Bronevetsky *et. al.*: C^3 (SC06, ASPLOS04, PPoPP03)
  - Zheng *et. al.*: Ftc-charm++ (Cluster04)

- Message logging
  - Agrabia *et. al.*: Starfish (Cluster03)
  - Bouteiller *et. al.*: Mpich-v (Int. Journal of High Perf. Comp. 06)

- Replication-based Fault Tolerance
  - Abawajy *et. al.* (IPDPS04)
Motivation and Introduction
Fault Tolerance System Design
Implementation of the System
Experimental Evaluation
Related Work
Conclusion
Conclusion

- Reduction object represents the state of the system
- Our FTS has very low overhead and effectively recovers from failures
- Different designs can be implemented using Robj.
- Our system outperforms Hadoop both in absence and presence of failures
Thanks