Interruptible Tasks: Treating Memory Pressure as Interrupts for Highly Scalable Data-Parallel Programs

Lu Fang ¹, Khanh Nguyen ¹, Guoqing(Harry) Xu ¹, Brian Demsky ¹, Shan Lu ²

¹University of California, Irvine
²University of Chicago

SOSP’15, October 7, 2015, Monterey, California, USA
Motivation

Data-parallel system

- Input data are divided into independent partitions
- Many popular big data systems

![Hadoop](image1.png)  ![Spark](image2.png)  ![Hive](image3.png)  ![Mahout](image4.png)
Motivation

Data-parallel system

- Input data are divided into independent partitions
- Many popular big data systems

⚠ Memory pressure on single nodes

Our study

- Search “out of memory” and “data parallel” in StackOverflow
- We have collected 126 related problems
Memory pressure on individual nodes

- Executions push heap limit (using managed language)
- Data-parallel systems struggle for memory

![Graph showing memory consumption, execution time, heap size, and OutOfMemoryError point. The graph indicates a long and useless GC.]
Memory Pressure in the Real World

Memory pressure on individual nodes

- Executions push heap limit (using managed language)
- Data-parallel systems struggle for memory

![Graph showing memory consumption, heap size, and execution time with OutOfMemoryError point and Long and useless GC]

CRASH OutOfMemory Error
Memory Pressure in the Real World

Memory pressure on individual nodes

- Executions push heap limit (using managed language)
- Data-parallel systems struggle for memory

---

CRASH

OutOfMemory Error

SLOW

Huge GC effort

Lu Fang et al.

Interruptible Tasks

SOSP’15, October 7, 2015
Root Cause 1: Hot Keys

Key-value pairs
Root Cause 1: Hot Keys

Key-value pairs

Popular keys have many associated values
Root Cause 1: Hot Keys

Key-value pairs

Popular keys have many associated values

Case study (from StackOverflow)

- Process StackOverflow posts
- Long and popular posts
- Many tasks process long and popular posts
Root Cause 2: Large Intermediate Results

Temporary data structures
Temporary data structures

Case study (from StackOverflow)

- Use NLP library to process customers’ review
- Some reviews are quite long
- NLP library creates giant temporary data structures for long reviews
Existing Solutions

More memory? Not really!

- Data double in size every two years, [http://goo.gl/tM92i0]
- Memory double in size every three years, [http://goo.gl/50Rrgk]
Existing Solutions

More memory? Not really!
- Data double in size every two years, [http://goo.gl/tM92i0]
- Memory double in size every three years, [http://goo.gl/50Rrgk]

Application-level solutions
- Configuration tuning
- Skew fixing
Existing Solutions

More memory? Not really!

- Data double in size every two years, [http://goo.gl/tM92i0]
- Memory double in size every three years, [http://goo.gl/50Rrgk]

Application-level solutions

- Configuration tuning
- Skew fixing

System-level solutions

- Cluster-wide resource manager, such as YARN
Existing Solutions

More memory? Not really!

▶ Data double in size every **two** years, [http://goo.gl/tM92i0]
▶ Memory double in size every **three** years, [http://goo.gl/50Rrgk]

Application-level solutions

▶ Configuration tuning
▶ Skew fixing

System-level solutions

▶ Cluster-wide resource manager, such as YARN

We need a **systematic and effective** solution!
**Interruptible Task:** *treat memory pressure as interrupt*

*Dynamically change parallelism degree*
Why Does Our Technique Help

Program starts with multiple tasks

Heap size

Execution time

Memory consumption

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory
Why Does Our Technique Help

<table>
<thead>
<tr>
<th>Task</th>
<th>Consumed Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Program pushes heap limit
Why Does Our Technique Help

Long and useless GC

Heap size

Execution time

Memory consumption

Task

Consumed Memory

Task

Consumed Memory

Task

Consumed Memory

Task

Consumed Memory

Lu Fang et al. Interruptible Tasks SOSP'15, October 7, 2015 8 / 25
Why Does Our Technique Help

Memory consumption vs Execution time graph with highlighted "OutOfMemory Error" point.
Why Does Our Technique Help

Long and useless GCs are detected

Heap size

Execution time

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory
Why Does Our Technique Help

Task
Consumed Memory

Killed
Long and useless GCs are detected, start interrupting tasks

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory
Why Does Our Technique Help

Release the memory, memory pressure is gone

Consumed Memory

Local Data Structures
Processed Input
Unprocessed Input
Output

Heap size
Execution time

Memory consumption

Task
Killed
Consumed Memory

Task
Killed
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory
Why Does Our Technique Help

- Task Consumed Memory
- Task Consumed Memory
- Task Consumed Memory
- Task Consumed Memory
- Task Consumed Memory
- Task Consumed Memory

Killed

- Killed
- Killed

Consumed Memory

- Local Data Structures
  - Released
  - Processed Input
  - Unprocessed Input
  - Output

Release the memory, memory pressure is gone

Execution time

Heap size

Memory consumption

Lu Fang et al.

Interruptible Tasks

SOSP’15, October 7, 2015

8 / 25
Why Does Our Technique Help

Release the memory, memory pressure is gone

Task
Consumed Memory
Task
Consumed Memory
Task
Consumed Memory
Task
Consumed Memory

Consumed Memory

Killed

Killed

Consumed Memory

Local Data Structures
Processed Input
Unprocessed Input
Output

Released
Released

Heap size
Execution time

Memory consumption
Why Does Our Technique Help

Release the memory, memory pressure is gone

- Local Data Structures
- Processed Input
- Unprocessed Input
- Output

- Released
- Released
- Kept in memory, can be serialized
Why Does Our Technique Help

Release the memory, memory pressure is gone

Local Data Structures
Processed Input
Unprocessed Input
Output

Consumed Memory

Task
Killed
Consumed Memory

Task
Killed
Consumed Memory

Task
Killed
Consumed Memory

Task

Final result: push out and released

Consumed Memory

Killed
Consumed Memory

Killed
Consumed Memory

Killed
Consumed Memory

Killed
Consumed Memory
Why Does Our Technique Help

Release the memory, memory pressure is gone

- Local Data Structures: Released
- Processed Input: Released
- Unprocessed Input: Kept in memory, can be serialized
- Output: Final result: push out and released
- Intermediate result: kept in memory, can be serialized
Why Does Our Technique Help

Program executes without memory pressure

Heap size

Execution time

Memory consumption

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory

Killed

Killed
Why Does Our Technique Help

If there is enough memory, increase parallelism degree

Memory consumption
Heap size
Execution time

Lu Fang et al.
Interruptible Tasks
SOSP'15, October 7, 2015 8 / 25
Challenges

How to expose semantics

How to interrupt/reactivate tasks
Challenges

How to expose semantics $\rightarrow$ a programming model

How to interrupt/reactivate tasks
Challenges

How to expose semantics $\rightarrow$ a programming model

How to interrupt/reactivate tasks $\rightarrow$ a runtime system
Challenges

How to expose semantics → a programming model

How to interrupt/reactivate tasks → a runtime system
A unified representation of input/output

- Separate processed and unprocessed input
- Specify how to serialize and deserialize
The Programming Model

A unified representation of input/output
- Separate processed and unprocessed input
- Specify how to serialize and deserialize

A definition of an interruptible task
- Safely interrupt tasks
- Specify the actions when interrupt happens
- Merge the intermediate results
Representing Input/Output as DataPartitions

- How to separate processed and unprocessed input
- How to serialize and deserialize the data

```
// The DataPartition abstract class
abstract class DataPartition {
  // Some fields and methods
  ...
  // A cursor points to the first unprocessed tuple
  int cursor;
  // Serialize the DataPartition
  abstract void serialize();
  // Deserialize the DataPartition
  abstract DataPartition deserialize();
}
```
Representing Input/Output as DataPartitions

- How to separate processed and unprocessed input
- How to serialize and deserialize the data

A cursor points to the first unprocessed tuple

```java
abstract class DataPartition {
    // Some fields and methods...
    int cursor;
    // Serialize the DataPartition
    abstract void serialize();
    // Deserialize the DataPartition
    abstract DataPartition deserialize();
}
```
Representing Input/Output as DataPartitions

- How to separate processed and unprocessed input
- How to serialize and deserialize the data

1. A cursor points to the first unprocessed tuple
2. Users implement serialize and deserialize methods

```
DataPartition Abstract Class

// The DataPartition abstract class
abstract class DataPartition {
    // Some fields and methods
    ...
    // A cursor points to the first unprocessed tuple
    int cursor;
    // Serialize the DataPartition
    abstract void serialize();
    // Deserialize the DataPartition
    abstract DataPartition deserialize();
}
```
Defining an ITask

- What actions should be taken when interrupt happens
- How to safely interrupt a task

### ITask Abstract Class

```java
// The ITask interface in the library
abstract class ITask {
    // Some methods
    ...
    abstract void interrupt();
    boolean scaleLoop(DataPartition dp) {
        // Iterate dp, and process each tuple
        while (dp.hasNext()) {
            // If pressure occurs, interrupt
            if (HasMemoryPressure()) {
                interrupt();
                return false;
            }
            process();
        }
    }
}
```
Defining an ITask

- What actions should be taken when interrupt happens
- How to safely interrupt a task

1. In interrupt, we define how to deal with partial results

```java
abstract class ITask {
    // Some methods
    abstract void interrupt();
    boolean scaleLoop(DataPartition dp) {
        // Iterate dp, and process each tuple
        while (dp.hasNext()) {
            // If pressure occurs, interrupt
            if (HasMemoryPressure()) {
                interrupt();
                return false;
            }
            process();
        }
    }
}
```
Defining an ITask

- What actions should be taken when interrupt happens
- How to safely interrupt a task

1. In interrupt, we define how to deal with partial results

2. Tasks are always interrupted at the beginning in the scaleLoop

```java
abstract class ITask {
    // Some methods
    abstract void interrupt();
    boolean scaleLoop(DataPartition dp) {
        // Iterate dp, and process each tuple
        while (dp.hasNext()) {
            // If pressure occurs, interrupt
            if (HasMemoryPressure()) {
                interrupt();
                return false;
            }
            process();
        }
    }
}
```
Multiple Input for an ITask

- How to merge intermediate results

MITask Abstract Class

```java
// The MITask interface in the library
abstract class MITask extends ITask{
    // Most parts are the same as ITask
    ...
    // Only difference
    boolean scaleLoop(
        PartitionIterator<DataPartition> i) {
        // Iterate partitions through iterator
        while (i.hasNext()) {
            DataPartition dp = (DataPartition) i.next();
            // Iterate all the data tuples in this partition
            ...
        }
        return true;
    }
}
```
Multiple Input for an ITask

- How to merge intermediate results

scaleLoop takes a PartitionIterator as input

MITask Abstract Class

```java
abstract class MITask extends ITask{
    // Most parts are the same as ITask
    ...
    // Only difference
    boolean scaleLoop(
        PartitionIterator<DataPartition> i) {
        // Iterate partitions through iterator
        while (i.hasNext()) {
            DataPartition dp = (DataPartition) i.next();
            // Iterate all the data tuples in this partition
            ...
            return true;
        }
    }
}
```
ITask WordCount on Hyracks

MapOperator

class MapOperator extends ITask
    implements HyracksOperator {
        void interrupt() {
            // Push out final
            // results to shuffling
            ... 
        }
        // Some other fields and methods
        ...
    }

Lu Fang et al.
Interruptible Tasks
SOSP’15, October 7, 2015 14 / 25
RefCount WordCount on Hyracks

Map Operator

Merge Operator

Reduce Operator

Final

ReduceOperator

class ReduceOperator extends ITask
    implements HyracksOperator {
        void interrupt() {
            // Tag the results;
            // Output as intermediate
            // results
            ...
        }
        // Some other fields and methods
        ...
    }
class MergeTask extends MITask {
    void interrupt() {
        // Tag the results;
        // Output as intermediate results
    }
    // Some other fields and methods
    ...
}
Challenges

How to expose semantics $\rightarrow$ a programming model

How to interrupt/activate tasks $\rightarrow$ a runtime system
ITask Runtime System

Scheduler

Grow/Reduce

Monitor

Check

Reduce

Partition Manager

Data Partition

Data Partition

Data Partition

Memory

ITask Runtime System
ITask Runtime System

- **ITasks**
  - Scheduler
  - Monitor
- **Disk**
- **Input/Output**
- **Serialize/Deserialize**
- **Memory**
- **Data Partitions**
  - Data Partition
  - Data Partition
  - Data Partition
- **Grow/Reduce**
- **Reduce**

Lu Fang et al. Interruptible Tasks SOSP’15, October 7, 2015 16 / 25
We have implemented ITask on

- Hadoop 2.6.0
- Hyracks 0.2.14
Evaluation Environments

We have implemented ITask on

- Hadoop 2.6.0
- Hyracks 0.2.14

An 11-node Amazon EC2 cluster

- Each machine: 8 cores, 15GB, 80GB*2 SSD
Experiments on Hadoop

Goal

- Show the effectiveness on real-world problems
Experiments on Hadoop

Goal

- Show the **effectiveness** on real-world problems

Benchmarks

- Original: five real-world programs collected from Stack Overflow
- RFix: apply the fixes recommended on websites
- ITask: apply ITask on original programs

<table>
<thead>
<tr>
<th>Name</th>
<th>Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map-Side Aggregation (MSA)</td>
<td>Stack Overflow Full Dump</td>
</tr>
<tr>
<td>In-Map Combiner (IMC)</td>
<td>Wikipedia Full Dump</td>
</tr>
<tr>
<td>Inverted-Index Building (IIB)</td>
<td>Wikipedia Full Dump</td>
</tr>
<tr>
<td>Word Cooccurrence Matrix (WCM)</td>
<td>Wikipedia Full Dump</td>
</tr>
<tr>
<td>Customer Review Processing (CRP)</td>
<td>Wikipedia Sample Dump</td>
</tr>
</tbody>
</table>
Improvements

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Original Time</th>
<th>RFix Time</th>
<th>ITask Time</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSA</td>
<td>1047 (crashed)</td>
<td>48</td>
<td>72</td>
<td>-33.3%</td>
</tr>
<tr>
<td>IMC</td>
<td>5200 (crashed)</td>
<td>337</td>
<td>238</td>
<td>41.6%</td>
</tr>
<tr>
<td>IIB</td>
<td>1322 (crashed)</td>
<td>2568</td>
<td>1210</td>
<td>112.2%</td>
</tr>
<tr>
<td>WCM</td>
<td>2643 (crashed)</td>
<td>2151</td>
<td>1287</td>
<td>67.1%</td>
</tr>
<tr>
<td>CRP</td>
<td>567 (crashed)</td>
<td>6761</td>
<td>2001</td>
<td>237.9%</td>
</tr>
</tbody>
</table>

- With ITask, all programs survive memory pressure
- On average, ITask versions are 62.5% faster than RFix
Experiments on Hyracks

Goal

- Show the improvements on performance
- Show the improvements on scalability

<table>
<thead>
<tr>
<th>Name</th>
<th>Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordCount (WC)</td>
<td>Yahoo Web Map and Its Subgraphs</td>
</tr>
<tr>
<td>Heap Sort (HS)</td>
<td>Yahoo Web Map and Its Subgraphs</td>
</tr>
<tr>
<td>Inverted Index (II)</td>
<td>Yahoo Web Map and Its Subgraphs</td>
</tr>
<tr>
<td>Hash Join (HJ)</td>
<td>TPC-H Data</td>
</tr>
<tr>
<td>Group By (GR)</td>
<td>TPC-H Data</td>
</tr>
</tbody>
</table>
Experiments on Hyracks

Goal
- Show the improvements on **performance**
- Show the improvements on **scalability**

Benchmarks
- **Original**: five hand-optimized applications from repository
- **ITask**: apply ITask on original programs

<table>
<thead>
<tr>
<th>Name</th>
<th>Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordCount (WC)</td>
<td>Yahoo Web Map and Its Subgraphs</td>
</tr>
<tr>
<td>Heap Sort (HS)</td>
<td>Yahoo Web Map and Its Subgraphs</td>
</tr>
<tr>
<td>Inverted Index (II)</td>
<td>Yahoo Web Map and Its Subgraphs</td>
</tr>
<tr>
<td>Hash Join (HJ)</td>
<td>TPC-H Data</td>
</tr>
<tr>
<td>Group By (GR)</td>
<td>TPC-H Data</td>
</tr>
</tbody>
</table>
## Configurations for best performance

<table>
<thead>
<tr>
<th>Name</th>
<th>Thread Number</th>
<th>Task Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordCount (WC)</td>
<td>2</td>
<td>32KB</td>
</tr>
<tr>
<td>Heap Sort (HS)</td>
<td>6</td>
<td>32KB</td>
</tr>
<tr>
<td>Inverted Index (II)</td>
<td>8</td>
<td>16KB</td>
</tr>
<tr>
<td>Hash Join (HJ)</td>
<td>8</td>
<td>32KB</td>
</tr>
<tr>
<td>Group By (GR)</td>
<td>6</td>
<td>16KB</td>
</tr>
</tbody>
</table>

## Configurations for best scalability

<table>
<thead>
<tr>
<th>Name</th>
<th>Thread Number</th>
<th>Task Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordCount (WC)</td>
<td>1</td>
<td>4KB</td>
</tr>
<tr>
<td>Heap Sort (HS)</td>
<td>1</td>
<td>4KB</td>
</tr>
<tr>
<td>Inverted Index (II)</td>
<td>1</td>
<td>4KB</td>
</tr>
<tr>
<td>Hash Join (HJ)</td>
<td>1</td>
<td>4KB</td>
</tr>
<tr>
<td>Group By (GR)</td>
<td>1</td>
<td>4KB</td>
</tr>
</tbody>
</table>
Improvements on Performance

On average, ITask is 34.4% faster
On average, ITask scales to $6.3 \times +$ larger datasets
Conclusions

A programming model + a runtime system

- Non-intrusive
- Easy to use
Conclusions

A programming model + a runtime system

▶ Non-intrusive
▶ Easy to use

First systematic approach

▶ Help data-parallel tasks survive memory pressure

ITask improves performance and scalability

▶ On Hadoop, ITask is 62.5% faster
▶ On Hyracks, ITask is 34.4% faster
▶ ITask helps programs scale to 6.3× larger datasets
Thank You

Q & A