Detecting and Fixing Memory-Related Performance Problems in Managed Languages

Lu Fang

Committee: Prof. Guoqing Xu (Chair), Prof. Alex Nicolau, Prof. Brian Demsky

University of California, Irvine

May 26, 2017, Irvine, CA, USA
Internet Explorer has stopped working

Windows is checking for a solution to the problem...
Performance Problems in Real World

Android SDK: Resolving error markers

Initializing Java Tooling

Configuring classpath containers

Android SDK Content Loader (Waiting)

Android SDK: Resolving error markers (Waiting)

Android SDK: Resolving error markers (Waiting)

Android SDK: Resolving error markers (Waiting)
Many distributed systems, such as Spark, Hadoop, also suffer from performance problems.

`java.lang.OutOfMemoryError`: Java heap space
Commonly exist in real world applications

- Single-machine apps, such as Eclipse, IE
- Traditional databases, web servers, such as MySQL, Tomcat
- Big Data systems, such as Hadoop, Spark
Performance Problems

Commonly exist in real world applications

▶ Single-machine apps, such as Eclipse, IE
▶ Traditional databases, web servers, such as MySQL, Tomcat
▶ Big Data systems, such as Hadoop, Spark

Further exacerbated by managed languages

▶ Such as Java, C#
▶ Big overhead introduced by automatic memory management
Performance Problems

Commonly exist in real world applications

- Single-machine apps, such as Eclipse, IE
- Traditional databases, web servers, such as MySQL, Tomcat
- Big Data systems, such as Hadoop, Spark

Further exacerbated by managed languages

- Such as Java, C#
- Big overhead introduced by automatic memory management

Cannot be optimized by compilers

- Cannot understand the deep semantics
- Cannot guarantee the correctness
Performance Problems

Difficult to find, especially during development

- Invisible effect
- Often escape to production runs

Can lead to severe problems
- Scalability reductions
- Programs hang and crash
- Financial losses
Performance Problems

Difficult to find, especially during development
- Invisible effect
- Often escape to production runs

Difficult to fix
- Large systems are complicated
- Enough diagnostic information is necessary
- Problems may be located deeply in systems
Performance Problems

Difficult to find, especially during development

- Invisible effect
- Often escape to production runs

Difficult to fix

- Large systems are complicated
- Enough diagnostic information is necessary
- Problems may be located deeply in systems

Can lead to severe problems

- Scalability reductions
- Programs hang and crash
- Financial losses
Many solutions are proposed

- Pattern-based
- Mining-based
- Learning-based
Existing Solutions

Many solutions are proposed

- Pattern-based
- Mining-based
- Learning-based

Most are *postmortem* debugging techniques

- Require user logs/input to trigger bugs
- Bugs already escape to production runs
Drawbacks in Existing Works

- Lacking a general way to describe problems

- Cannot detect problems under small workload

- Lacking a systematic approach to tune memory usage in data-intensive systems
Drawbacks in Existing Works → Our Solutions

- Lacking a general way to describe problems
  → Instrumentation Specification Language (ISL)

- Cannot detect problems under small workload

- Lacking a systematic approach to tune memory usage in data-intensive systems
Drawbacks in Existing Works → Our Solutions

- Lacking a general way to describe problems
  → Instrumentation Specification Language (ISL)

- Cannot detect problems under small workload
  → PerfBlower

- Lacking a systematic approach to tune memory usage in data-intensive systems
Drawbacks in Existing Works → Our Solutions

- Lacking a general way to describe problems
  → Instrumentation Specification Language (ISL)

- Cannot detect problems under small workload
  → PerfBlower

- Lacking a systematic approach to tune memory usage in data-intensive systems
  → ITask
Lu Fang, Liang Dou, Guoqing Xu

PerfBlower: Quickly Detecting Memory-Related Performance Problems via Amplification

ECOOP’15
Motivation 1: an easy way to develop new detectors
• Motivation 1: an easy way to develop new detectors

• Motivation 2: detect the problems with small effects
Focus on problems with observable heap symptoms

Users define symptoms/counter-evidence in events

Two important actions: *amplify* and *deamplify*
amplify: increases the penalty
amplify: increases the penalty

deamplify: resets the penalty
Amplification and Deamplification

amplify: increases the penalty
deamplify: resets the penalty

Virtual space overhead (VSO)

\[ VSO = \frac{\text{Sum}_{\text{penalty}} + \text{Size}_{\text{live heap}}}{\text{Size}_{\text{live heap}}} \]

- Reflects the severity on 2 dimensions: Time and Size
Detecting Leaking Object Arrays

Context TypeContext {
    type = "java.lang.Object[]";
}
History UseHistory {
    type = "boolean";
    size = 10;
}
Partition AllPartition {
    kind = all;
    history = UseHistory;
}
TObject TrackedObject {
    include = TypeContext;
    partition = AllPartition;
    instance boolean useFlag = false;
}
Event on_rw(Object o, Field f, Word w1, Word w2) {
    o.useFlag = true;
    deamplify(o);
}
Event on_reachedOnce(Object o) {
    UseHistory h = getHistory(o);
    h.update(o.useFlag);
    if (h.isFull() && !h.contains(true)) amplify(o);
    o.useFlag = false;
}
An ISL Program Example

1. Context defines the type
2. History of partition instance
3. Heap partitioning
4. Tracked objects

Detecting Leaking Object Arrays

```java
Context TypeContext {
    type = 'java.lang.Object[]';
}
History UseHistory {
    type = 'boolean';
    size = 10;
}
Partition AllPartition {
    kind = all;
    history = UseHistory;
}
TObject TrackedObject {
    include = TypeContext;
    partition = AllPartition;
    instance boolean useFlag = false;
}
Event on_rw(Object o, Field f, Word w1, Word w2) {
    o.useFlag = true;
    deamplify(o);
}
Event on_reachedOnce(Object o) {
    UseHistory h = getHistory(o);
    h.update(o.useFlag);
    if (h.isFull() && !h.contains(true)) amplify(o);
    o.useFlag = false;
}
```
An ISL Program Example

1. Context defines the type
2. History of partition instance
3. Heap partitioning
4. Tracked objects
5. The actions on events

Detecting Leaking Object Arrays

```java
Context TypeContext {
    type = 'java.lang.Object[]';
}
History UseHistory {
    type = 'boolean';
    size = 10;
}
Partition AllPartition {
    kind = all;
    history = UseHistory;
}
TObject TrackedObject {
    include = TypeContext;
    partition = AllPartition;
    instance boolean useFlag = false;
}
Event on_rw(Object o, Field f, Word w1, Word w2) {
    o.useFlag = true;
    deamplify(o);
}
Event on_reachedOnce(Object o) {
    UseHistory h = getHistory(o);
    h.update(o.useFlag);
    if (h.isFull() && !h.contains(true)) amplify(o);
    o.useFlag = false;
}
```
A general performance testing framework

Supports ISL

Can capture problems with small effects

Reports reference path to problematic objects
Object leak is referenced by array

Leak is reference by whom?

```java
Object[] array = new Object[10];

// Allocation site 1, creating the leak.
Object leak = new Object();

// Object leak is referenced by array
array[0] = leak;

// Keep using Object leak
...

// ... Never use leak again.
// However, leak is referenced by array,
// GC cannot reclaim object leak.
```
Heap Reference Path

1. Object *leak* is referenced by *array*

2. Knowing allocation site 1 is not enough

---

**Leak is reference by whom?**

```java
Object[] array = new Object[10];

// Allocation site 1, creating the leak.
Object leak = new Object();

// Object leak is referenced by array
array[0] = leak;

// Keep using Object leak
...

// ... Never use leak again.
// However, leak is referenced by array,
// GC cannot reclaim object leak.
```
Object *leak* is referenced by *array*

Knowing allocation site 1 is not enough

Key point: *array* keeps a reference to *leak*, which can be shown by *leak’s heap reference path*

---

**Leak is reference by whom?**

```java
Object[] array = new Object[10];

// Allocation site 1, creating the leak.
Object leak = new Object();

// Object leak is referenced by array
array[0] = leak;

// Keep using Object leak
...

// ... Never use leak again.
// However, leak is referenced by array,
// GC cannot reclaim object leak.
```
Mirroring the reference path

Original Objects

Object obj

Object[] elements

Stack s

Mirror obj'

Mirror elements'

Mirror s'

Stack stack = new stack;

// Allocation site 1, creating the leak.
Object obj = new Object();

// stack.elements[0] = leak
stack.push();

// Keep using Object leak
...

// ... Never use obj again
// However, leak is referenced by stack,
// GC cannot reclaim object leak.

Mirroring Ref. Path
Three detectors

- Memory leak amplifier
- Under-utilized container amplifier
- Over-populated container amplifier

DaCapo benchmarks with 500MB heap
Memory Leak Amplifier

VSOs caused by confirmed memory leaks

Basic VSOs

VSO is large ➔ The program is likely to have leaks

Programs with confirmed unknown leaks
VSOs caused by confirmed under-utilized containers

Basic VSOs

VSO is large ➔ The program is very likely to have UUCs

Programs with confirmed unknown UUCs
VSOs caused by confirmed over-populated containers

Basic VSOs

VSO is large $\rightarrow$ The program is very likely to have OPCs

Programs with confirmed unknown OPCs

antlr  bloat  eclipse  fop  luindex  lusearch  pmd  xalan  hsqldb  jython
## Performance Improvements

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Space Reduction</th>
<th>Time Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>xalan-leak</td>
<td>25.4%</td>
<td>14.6%</td>
</tr>
<tr>
<td>jython-leak</td>
<td>24.3%</td>
<td>7.4%</td>
</tr>
<tr>
<td>hsqldb-leak</td>
<td>15.6%</td>
<td>3.1%</td>
</tr>
<tr>
<td>xalan-UUC</td>
<td>5.4%</td>
<td>34.1%</td>
</tr>
<tr>
<td>jython-UUC</td>
<td>19.1%</td>
<td>1.1%</td>
</tr>
<tr>
<td>hsqldb-UUC</td>
<td>17.4%</td>
<td>0.7%</td>
</tr>
<tr>
<td>hsqldb-OPC</td>
<td>14.9%</td>
<td>2.9%</td>
</tr>
</tbody>
</table>
The Effectiveness of PerfBlower

VSOs indicate the existence of problems

- 8 unknown problems are detected
- All reports contain useful diagnostic information

Low overhead

- Space overheads are 1.23–1.25×
- Time overheads are 2.39–2.74×
Fixing performance problems is hard

- Enough information is necessary
- Have to understand the logic of the system
- The problem exists deeply in the system

Memory pressure

- A common performance problem in data-parallel systems
Lu Fang, Khanh Nguyen, Guoqing Xu, Brian Demsky, Shan Lu

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

SOSP’15
Data-parallel system

- Input data are divided into independent partitions
- Many popular big data systems
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, and heap size with OutOfMemoryError point and Long and useless GC points.](Image)
Memory pressure on individual nodes

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

---

Memory consumption
Execution time
Heap size
OutOfMemoryError point
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

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

**CRASH** OutOfMemory Error

**SLOW** Huge GC effort
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
Temporary data structures
Root Cause 2: Large Intermediate Results

Temporary data structures

Case study (from StackOverflow)

- Use NLP library to process customers’ reviews
- 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](http://goo.gl/tM92i0)
- Memory double in size every **three** years, [http://goo.gl/50Rrgk](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 \textcolor{red}{two} years, [http://goo.gl/tM92i0]
- Memory double in size every \textcolor{red}{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

- Task
  - Consumed Memory

- Task
  - Consumed Memory

- Task
  - Consumed Memory

- Task
  - Consumed Memory
Why Does Our Technique Help

- Program pushes heap limit

Memory consumption

Heap size

Execution time

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory

Lu Fang (UC Irvine)
Why Does Our Technique Help

Execution time

Heap size

Memory consumption

Long and useless GC

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory

Lu Fang (UC Irvine) Final Defense May 26, 2017 30 / 51
Why Does Our Technique Help

Memory consumption

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory

STOP OutOfMemory Error

Heap size

Execution time

Lu Fang (UC Irvine)
Why Does Our Technique Help

- Task
- Consumed Memory

Long and useless GCs are detected

Heap size

Execution time

Memory consumption
Why Does Our Technique Help

Long and useless GCs are detected, start interrupting tasks

Memory consumption vs. Execution time

Heap size
Why Does Our Technique Help

- Memory consumption
- Execution time
- Heap size

Release the memory, memory pressure is gone

Consumed Memory:
- Local Data Structures
- Processed Input
- Unprocessed Input
- Output

Task
- Consumed Memory

Killed:
- Task
- Consumed Memory

Lu Fang (UC Irvine)
Why Does Our Technique Help

Release the memory, memory pressure is gone

Memory consumption

Heap size

Execution time

Consumed Memory

Killed

Local Data Structures

Processed Input

Unprocessed Input

Output

Consumed Memory

Killed

Released
Why Does Our Technique Help

Release the memory, memory pressure is gone

Task
Consumed Memory

Killed
Task
Consumed Memory

Killed
Task
Consumed Memory

Consumed Memory

Local Data Structures

Processed Input

Unprocessed Input

Output

Released

Released

Heap size

Memory consumption

Execution time

Lu Fang (UC Irvine)
Why Does Our Technique Help

Release the memory, memory pressure is gone

Heap size

Memory consumption

Execution time

Consumed Memory

Task

Killed

Consumed Memory

Task

Killed

Local Data Structures

Processed Input

Unprocessed Input

Output

Killed Consumed Memory

Consumed Memory

Task

Consumed Memory

Task

Consumed Memory

Task

Consumed Memory

Task

Consumed Memory

Released

Released

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

Release the memory, memory pressure is gone

- Task
  - Consumed Memory

- Task
  - Consumed Memory

- Task
  - Consumed Memory

- Task
  - Consumed Memory

Completed Memory

- Local Data Structures
  - Released
- Processed Input
  - Released
- Unprocessed Input
  - Kept in memory, can be serialized
- Output
  - Final result: push out and released
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

Killed

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory

Task
Consumed Memory

Killed

Lu Fang (UC Irvine)
Why Does Our Technique Help

If there is enough memory, increase parallelism degree

Memory consumption
Execution time
Heap size

Task
Consumed Memory

Killed
Consumed Memory

Killed
Consumed Memory

Newly created
Consumed Memory

Lu Fang (UC Irvine)
Final Defense
May 26, 2017
Making Task Interruptible Is Non-trivial

Interrupted

Task

Consumed Memory

Consumed Memory

Released or Kept in Memory?

Released or Kept in Memory?

Released or Kept in Memory?

Released or Kept in Memory?
Making Task Interruptible Is Non-trivial

Require Semantics
Challenges

How to expose semantics

How to interrupt/reactivate tasks
Challenges

How to expose semantics → a programming model

How to interrupt/reactivate tasks
Challenges

How to expose semantics → a programming model

How to interrupt/reactivate tasks → a runtime system
Challenges

How to expose semantics $\rightarrow$ a programming model

How to interrupt/reactivate tasks $\rightarrow$ a runtime system
The Programming Model

An ITask requires more semantics

- Separate processed and unprocessed input
- Specify how to serialize and deserialize
- Safely interrupt tasks
- Specify the actions when interrupt happens
- Merge the intermediate results
The Programming Model

An ITask requires more semantics
- Separate processed and unprocessed input
- Specify how to serialize and deserialize
- Safely interrupt tasks
- Specify the actions when interrupt happens
- Merge the intermediate results

A unified representation of input/output

A definition of an interruptible task
How to separate processed and unprocessed input

How to serialize and deserialize the data

**DataPartition Abstract Class**

```java
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

DataPartition Abstract Class

```java
// 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

1. A cursor points to the first unprocessed tuple

2. Users implement serialize and deserialize methods

```java
// 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

In interrupt, we define how to deal with partial results

```
// 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

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

```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();
        }
    }
}
```
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
    ...
    // The 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

1. `scaleLoop` takes a `PartitionIterator` as input

MITask Abstract Class

```java
// The MITask interface in the library
abstract class MITask extends ITask{
    // Most parts are the same as ITask
    ...
    // The 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;
    }
}
```
class MapOperator extends ITask implements HyracksOperator {
    void interrupt() {
        // Push out final
        // results to shuffling
        ...
    }
    // Some other fields and methods
    ...
}
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 → a programming model

How to interrupt/activate tasks → a runtime system
ITask Runtime System

Scheduler

Monitor

Partition Manager

Data Partition
Data Partition
Data Partition

Memory

Grow/Reduce

Check

Reduce
ITask Runtime System

ITasks

Memory

Scheduler

Monitor

Disk

Partition Manager

Data Partition

Input/Output

Serialize/Deserialize

Grow/Reduce

Reduce

ITask Runtime System
ITask Runtime System

ITasks

Scheduler

Monitor

Disk

Memory

Interrupt/Create

Input/Output

Serialize/Deserialize

Grow/Reduce

Reduce

Data Partition

Data Partition

Data Partition

Lu Fang (UC Irvine)

Final Defense

May 26, 2017
We have implemented ITask on

- Hadoop 2.6.0
- Hyracks 0.2.14
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>
<tr>
<td>Benchmark</td>
<td>Original Time</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>MSA</td>
<td>1047 (crashed)</td>
</tr>
<tr>
<td>IMC</td>
<td>5200 (crashed)</td>
</tr>
<tr>
<td>IIB</td>
<td>1322 (crashed)</td>
</tr>
<tr>
<td>WCM</td>
<td>2643 (crashed)</td>
</tr>
<tr>
<td>CRP</td>
<td>567 (crashed)</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**
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>
### Tuning Configurations for Original Programs

#### 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>
On average, ITask is 34.4% faster.
On average, ITask scales to $6.3 \times +$ larger datasets
The Effectiveness of ITask

ITask is practical
- it has helped 13 real-world applications survive memory problems

ITask improves performance and scalability
- On Hadoop, ITask is 62.5% faster
- On Hyracks, ITask is 34.4% faster
- ITask helps programs scale to $6.3 \times$ larger datasets

A programming model + a runtime system
- Non-intrusive
- Easy to use
Conclusions

First general technique to amplify problems

- A class of performance problems
- Reveals potential problems during testing

A general performance testing framework

- Includes a compiler and a runtime system
- Very practical

First systematic approach to address memory pressure

- Consists of a programming model and a runtime system
- Solves real-world problems
- Significantly improves data-parallel tasks’ performance and scalability
Future Works

Extend ISL

Add support into production JVMs

Consider more factors to improve test oracle

Instantiate ITask in more data-parallel systems
K. Nguyen, L. Fang, G. Xu, B. Demsky, S. Lu, S. Alamian, O. Mutlu
*Yak: A High-Performance Big-Data-Friendly Garbage Collector*
OSDI’16

Z. Zuo, L. Fang, S. Khoo, G. Xu, S. Lu
*Low-Overhead and Fully Automated Statistical Debugging with Abstraction Refinement*
OOPSLA’16

K. Nguyen, L. Fang, G. Xu, B. Demsky.
*Speculative Region-based Memory Management for Big Data Systems*
PLOS’15

L. Fang, K. Nguyen, G. Xu, B. Demsky, S. Lu
*Interruptible Tasks: Treating Memory Pressure As Interrupts for Highly Scalable Data-Parallel Programs*
SOSP’15

L. Fang, L. Dou, G. Xu
*PerfBlower: Quickly Detecting Memory-Related Performance Problems via Amplification*
ECOOP’15

K. Nguyen, K. Wang, Y. Bu, L. Fang, J. Hu, G. Xu
*Facade: A Compiler and Runtime for (Almost) Object-Bounded Big Data Applications*
ASPLOS’15
Q & A