Detecting and Fixing Memory-Related Performance Problems in Managed Languages

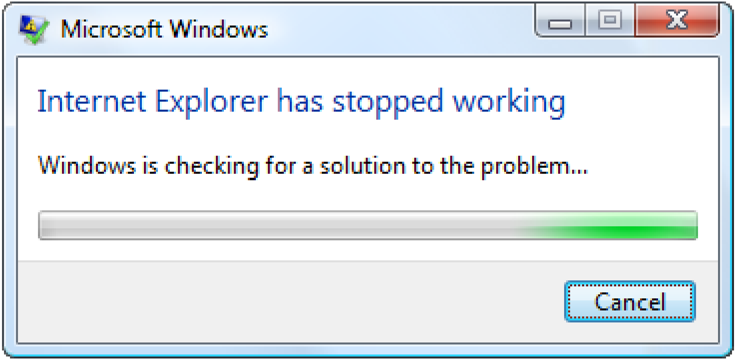
Lu Fang

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

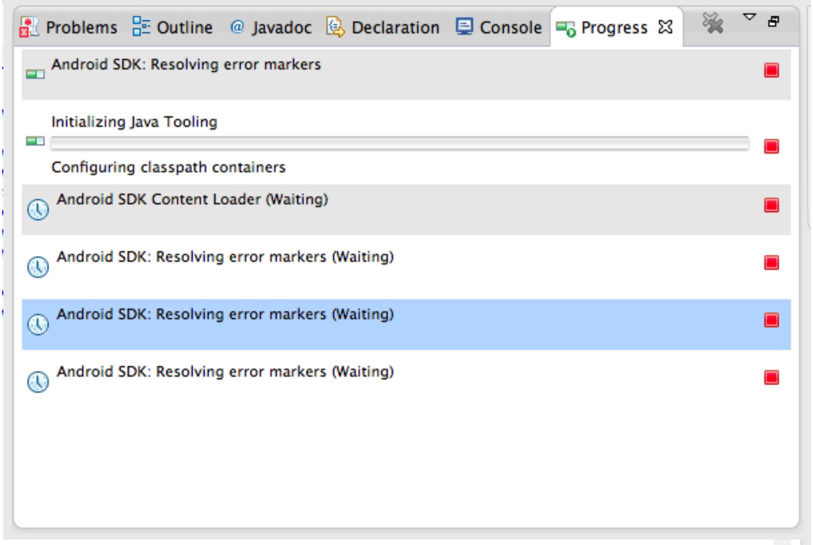
University of California, Irvine

May 26, 2017, Irvine, CA, USA

Performance Problems in Real World



Performance Problems in Real World




ERROR
503
Service unavailable.

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

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

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

Difficult to find, especially during development

- Invisible effect
- Often escape to production runs

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

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

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

UCIrvine

- 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

Amplification and Deamplification

`amplify`: increases the penalty

`deamplify`: resets the penalty

`amplify`: increases the penalty

`deamplify`: resets the penalty

Virtual space overhead (VSO)

- $VSO = \frac{Sum_{penalty} + Size_{live\_heap}}{Size_{live\_heap}}$
- Reflects the severity on 2 dementions: **Time** and **Size**

An ISL Program Example

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

```
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

```
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;
}
```

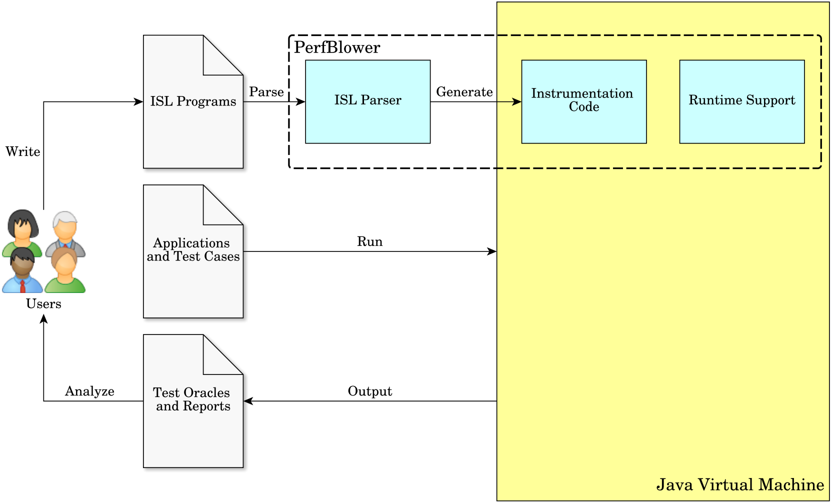
A general performance testing framework

Supports ISL

Can capture problems with small effects

Reports reference path to problematic objects

PerfBlower



Write
ISL Programs
Parse
PerfBlower
ISL Parser
Generate
Instrumentation Code
Runtime Support
Users
Applications and Test Cases
Run
Analyze
Test Oracles and Reports
Output
Java Virtual Machine

Heap Reference Path

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

Leak is reference by whom?

```
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?

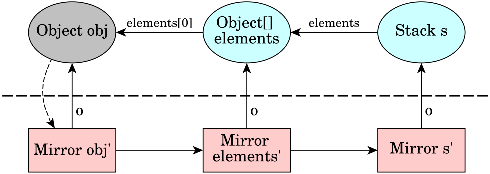
```
Object[] array = new Object[10];

// 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.
```

1. Object *leak* is referenced by *array*
2. Knowing allocation site 1 is not enough
3. 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.
```

Original Objects



Mirror Objects

Mirroring Ref. Path

```
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.
```

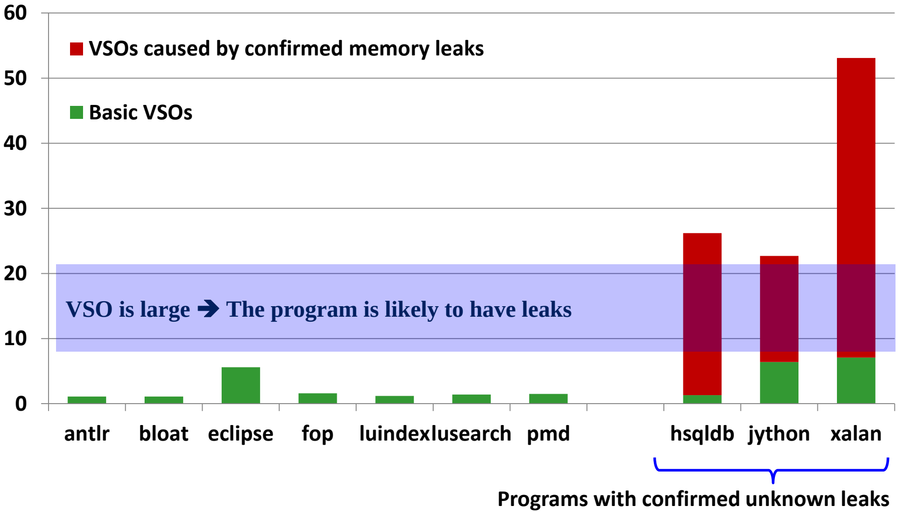
Three detectors

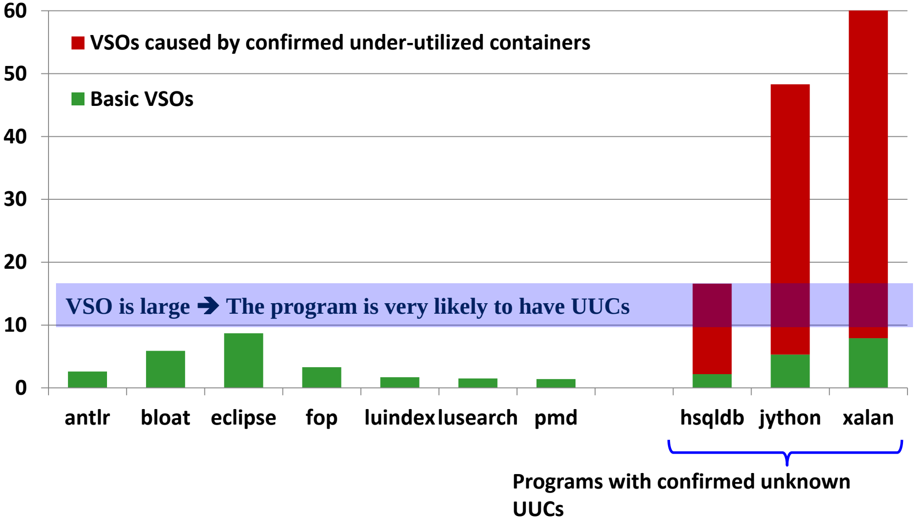
- Memory leak amplifier
- Under-utilized container amplifier
- Over-populated container amplifier

DaCapo benchmarks with 500MB heap

Memory Leak Amplifier

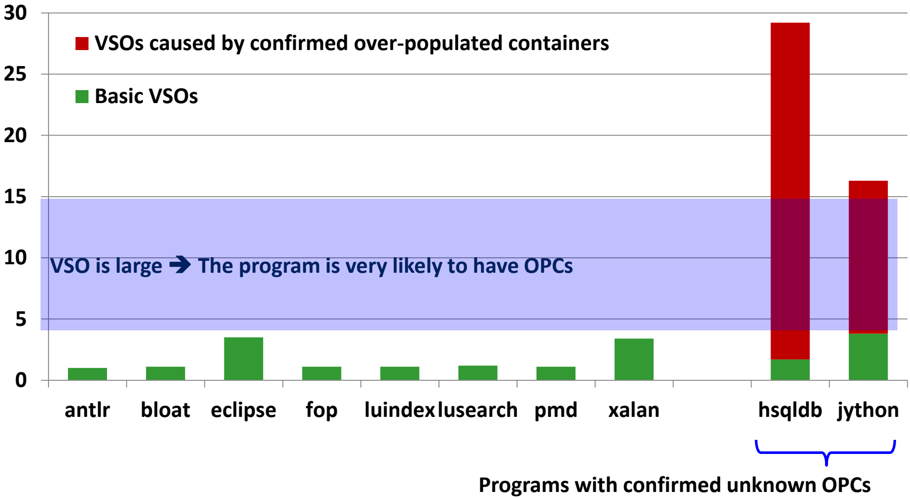
UCIrvine





60
50
40
30
20
10
0
VSOs caused by confirmed under-utilized containers
Basic VSOs
VSO is large ➔ The program is very likely to have UUCs
antlr
bloat
eclipse
fop
luindex
lusearch
pmd
hsqldb
jython
xalan
Programs with confirmed unknown UUCs

Over-Populated Container Amplifier



Performance Improvements

Benchmark	Space Reduction	Time Reduction
xalan-leak	25.4%	14.6%
jython-leak	24.3%	7.4%
hsqldb-leak	15.6%	3.1%
xalan-UUC	5.4%	34.1%
jython-UUC	19.1%	1.1%
hsqldb-UUC	17.4%	0.7%
hsqldb-OPC	14.9%	2.9%

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-paralle 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

Memory Pressure in Data-Parallel Systems

UCIRVINE

Data-parallel system

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

Memory Pressure in Data-Parallel Systems

Data-parallel system

- Input data are divided into <u>independent</u> partitions
- Many popular big data systems

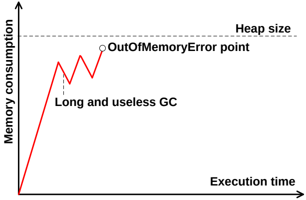
Memory pressure on single nodes

Our study

- Search "out of memory" and "data parallel" in StackOverflow
- We have collected 126 related problems

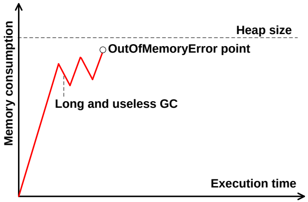
Memory pressure on individual nodes

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



Memory pressure on individual nodes

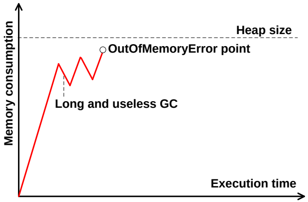
- Executions push heap limit (using managed language)
- Data-parallel systems struggle for memory



OutOfMemory Error

Memory pressure on individual nodes

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



OutOfMemory Error

Huge GC effort

Key-value pairs

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' reviews
- Some reviews are quite long
- NLP library creates giant temporary data structures for long reviews

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]

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

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

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

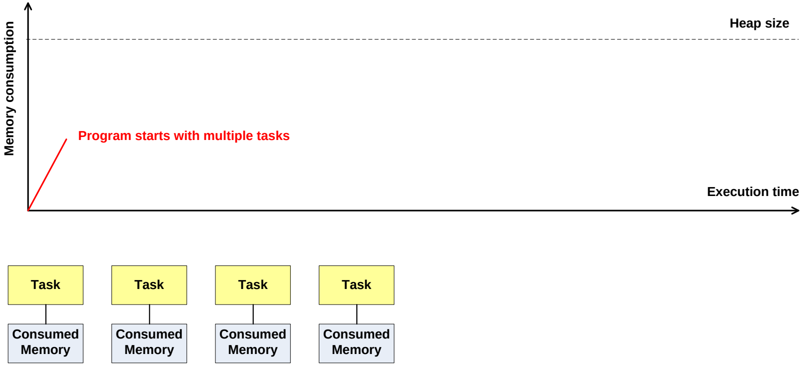
- Cluster-wide resource manager, such as YARN

We need a systematic and effective solution!

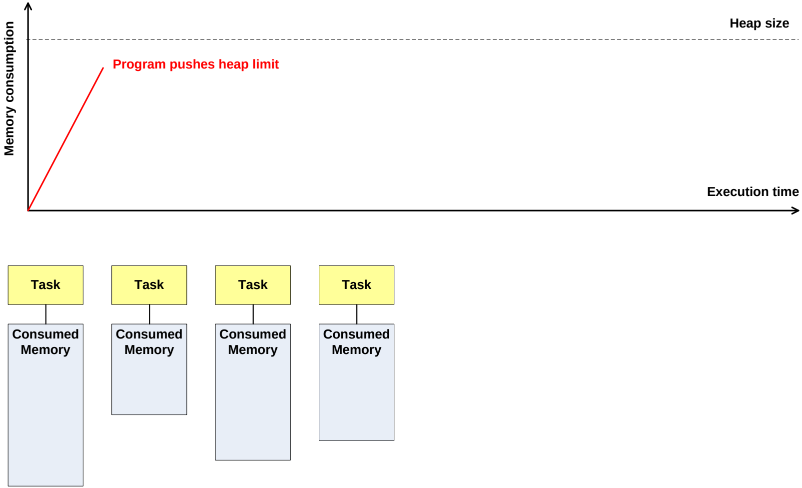
Interruptible **Task**: *treat memory pressure as interrupt*

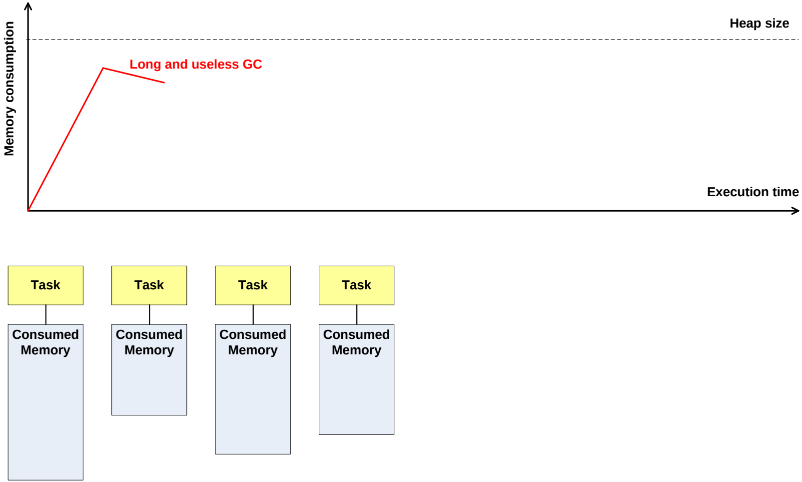
Dynamically change parallelism degree

Why Does Our Technique Help

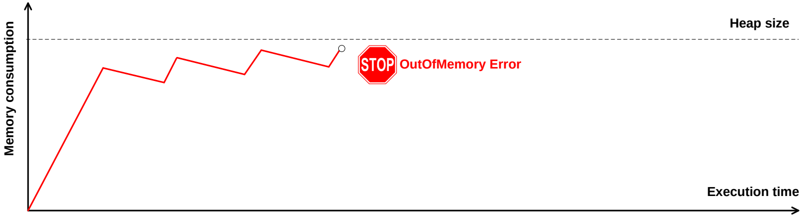


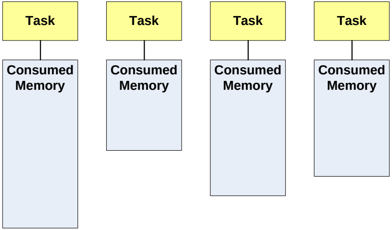
Why Does Our Technique Help



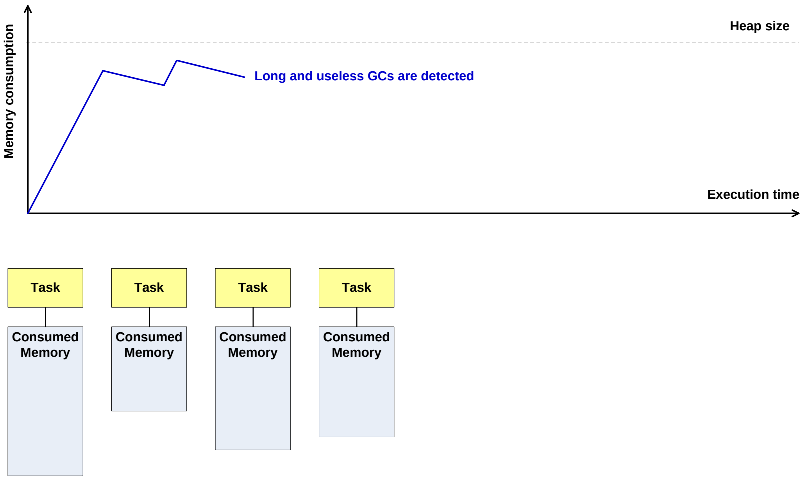

Memory consumption
Heap size
Long and useless GC
Execution time
Task
Consumed Memory
Task
Consumed Memory
Task
Consumed Memory
Task
Consumed Memory

Why Does Our Technique Help

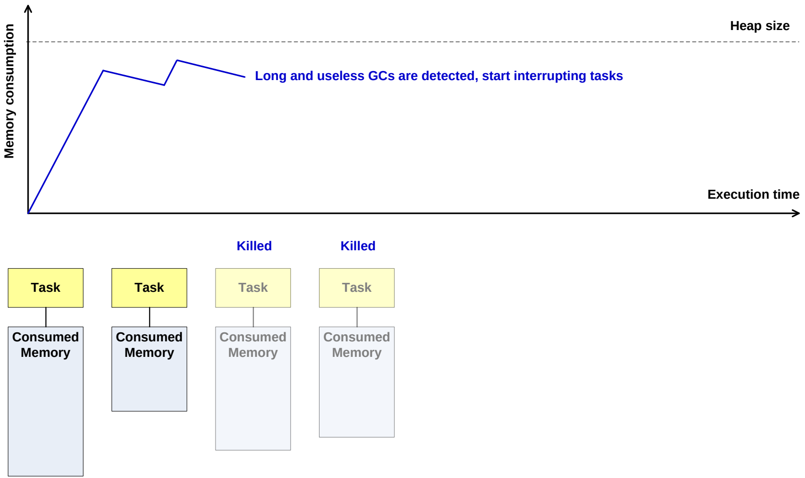




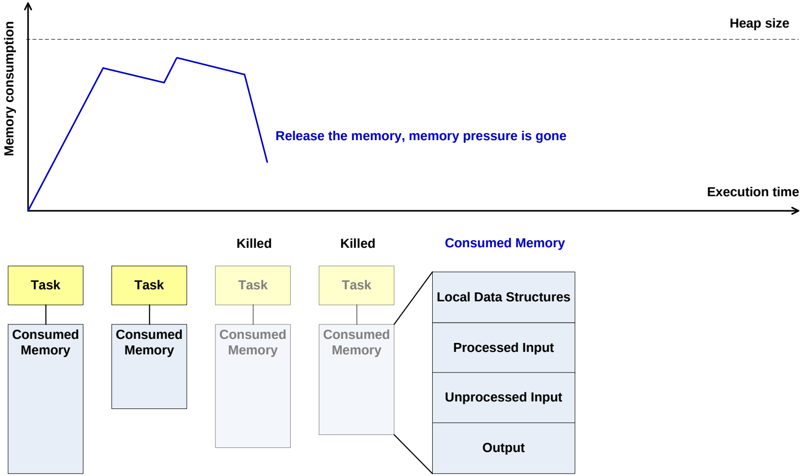
Why Does Our Technique Help



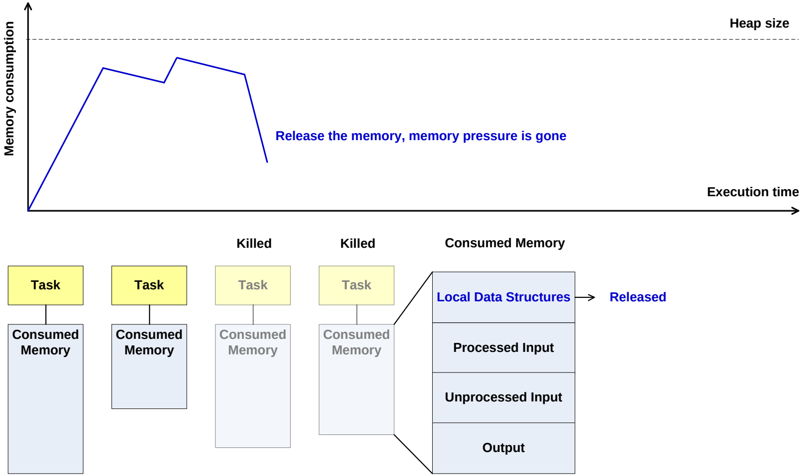
Why Does Our Technique Help



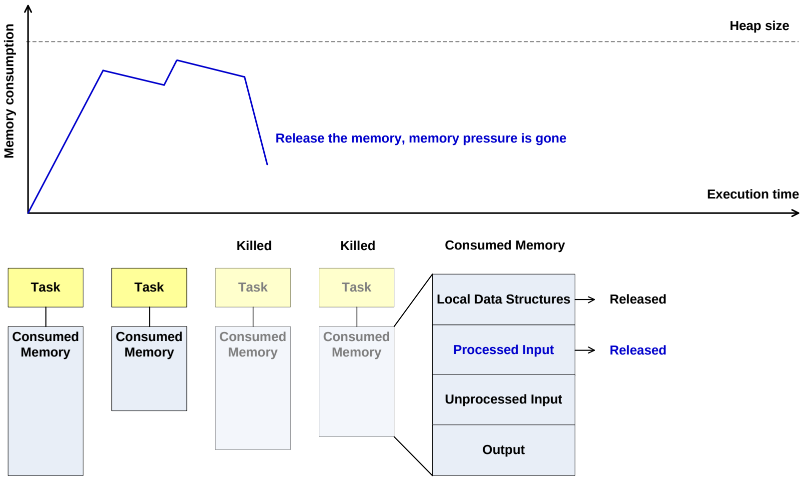
Why Does Our Technique Help



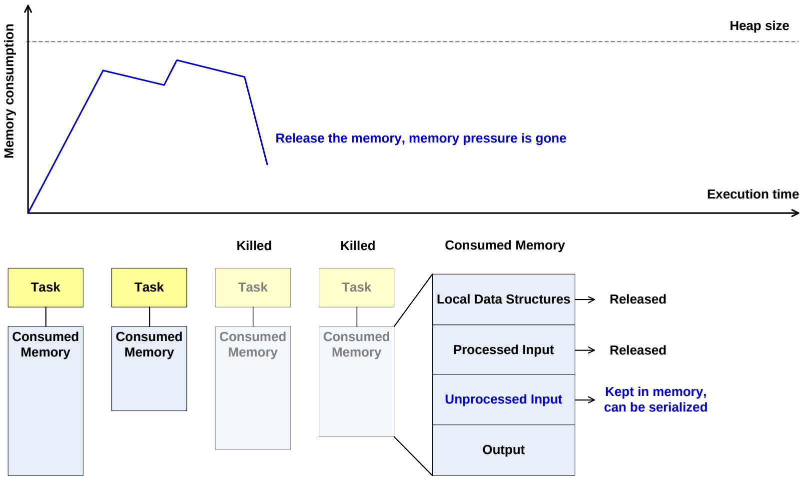
Why Does Our Technique Help



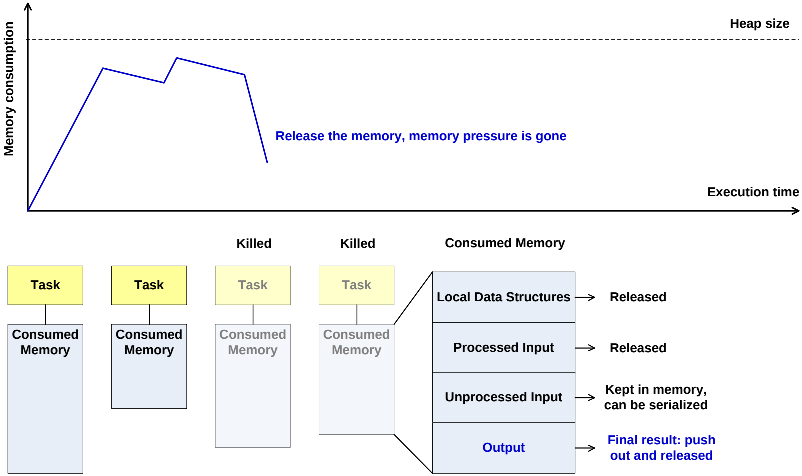
Why Does Our Technique Help



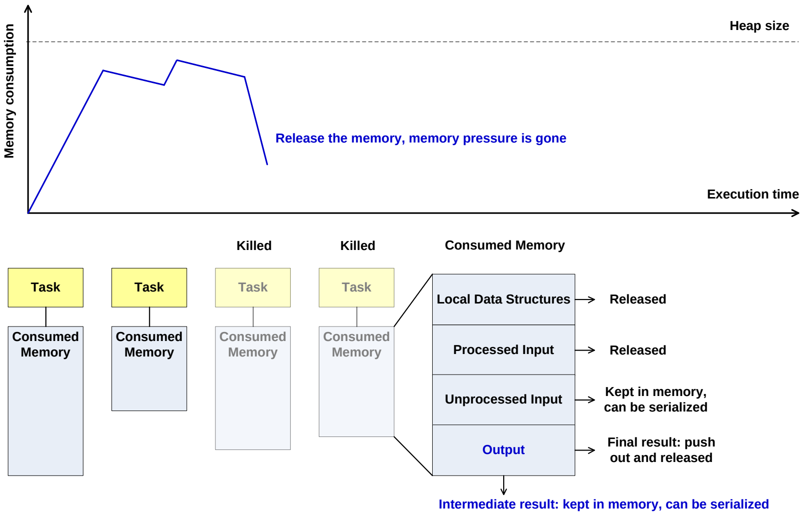
Why Does Our Technique Help



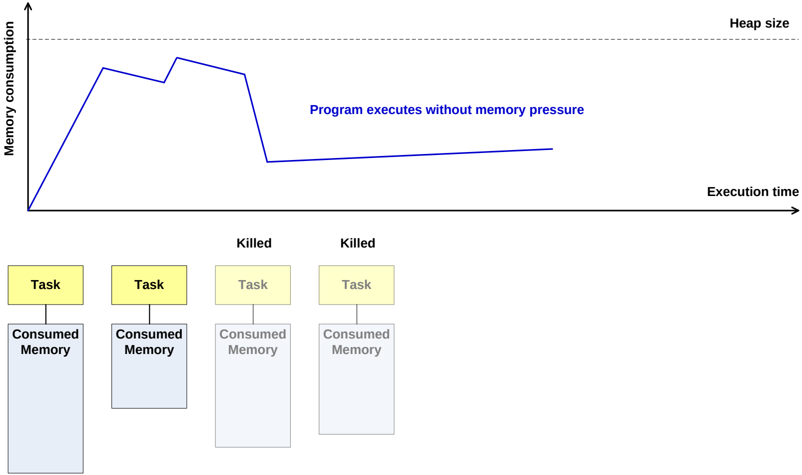
Why Does Our Technique Help



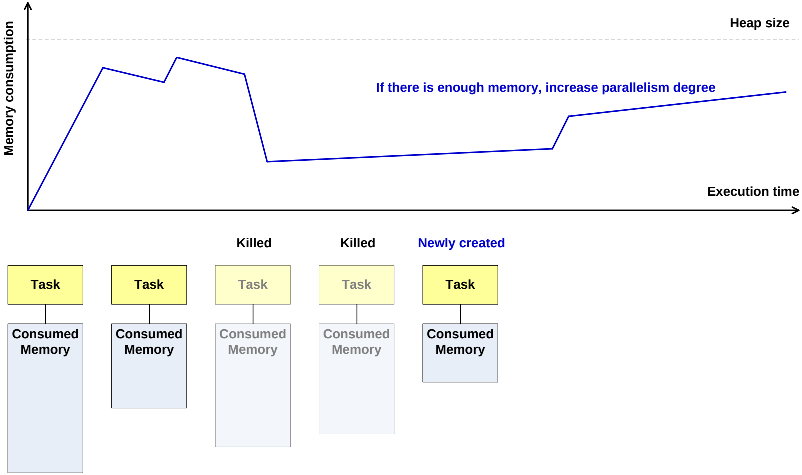
Why Does Our Technique Help



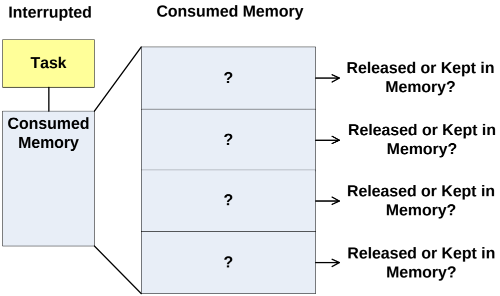
Why Does Our Technique Help

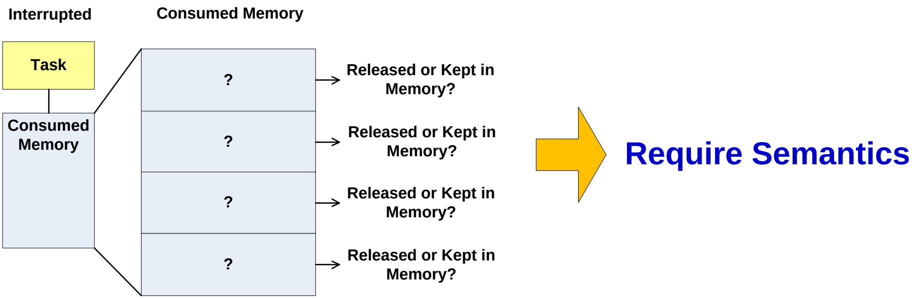


Why Does Our Technique Help



Making Task Interruptible Is Non-trivial




Interrupted
Consumed Memory
Task
Consumed Memory
?
?
?
?
Released or Kept in Memory?
Released or Kept in Memory?
Released or Kept in Memory?
Released or Kept in Memory?
Require Semantics

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 $\rightarrow$ a programming model

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

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

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

```
// 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();
}
```

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

1. A cursor points to the first unprocessed tuple

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();
}
```

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

```
// 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

ITask Abstract Class

```
// 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

ITask Abstract Class

```
// 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();
    }
  }
}
```

Multiple Input for an ITask

- How to merge intermediate results

MITask Abstract Class

```
// 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

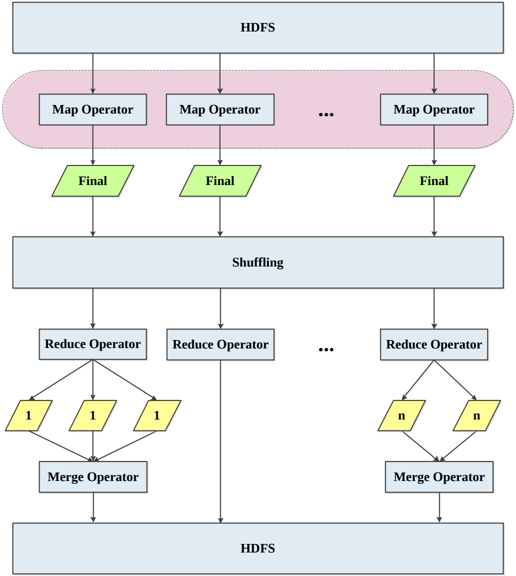
- How to merge intermediate results

1. scaleLoop takes a PartitionIterator as input

MITask Abstract Class

```
// 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;
  }
}
```

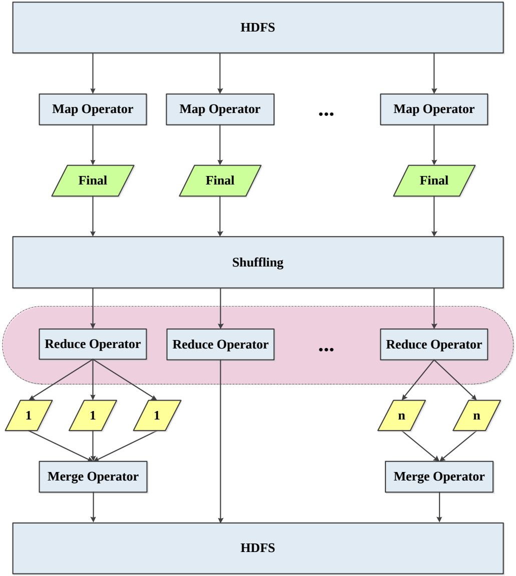
ITask WordCount on Hyracks



MapOperator

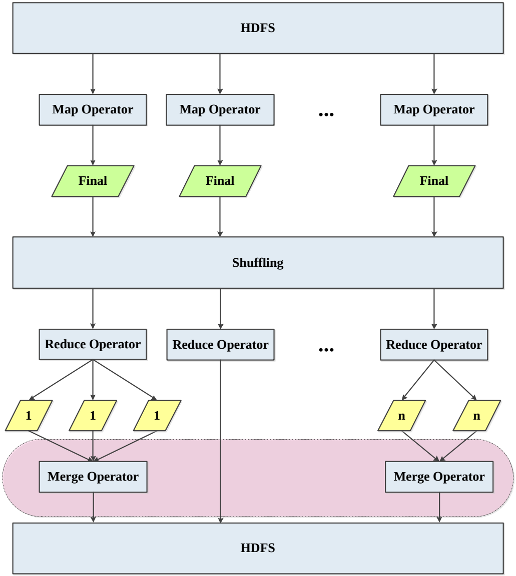
```
class MapOperator extends ITask
      implements HyracksOperator {
  void interrupt() {
    // Push out final
    // results to shuffling
    ...
  }
  // Some other fields and methods
  ...
}
```

ITask WordCount on Hyracks



ReduceOperator

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



MergeOperator

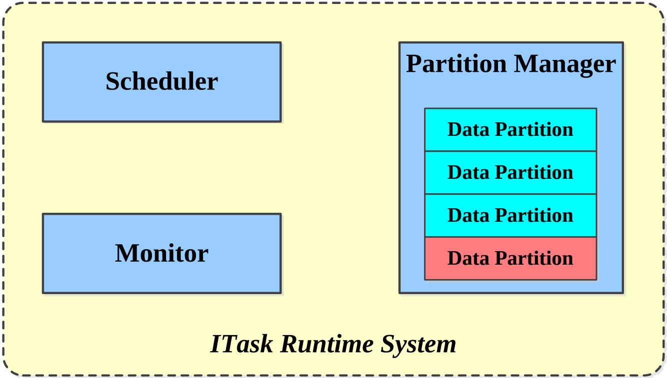
```
class MergeTask extends MITask {
  void interrupt() {
    // Tag the results;
    // Output as intermediate
    // results
  }
  // Some other fields and methods
  ...
}
```

Challenges

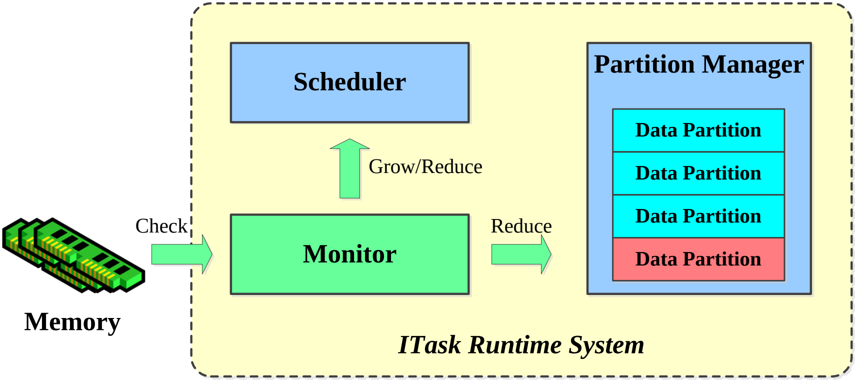
How to expose semantics $\rightarrow$ a programming model

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

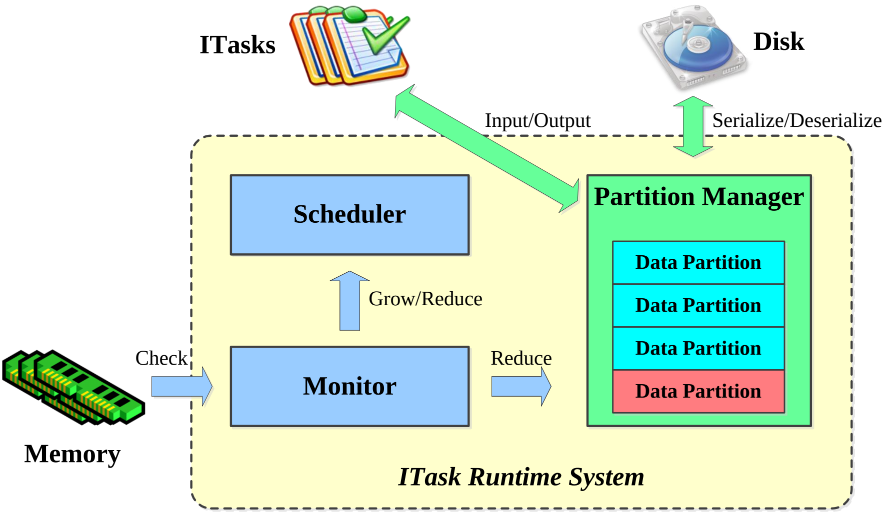
ITask Runtime System



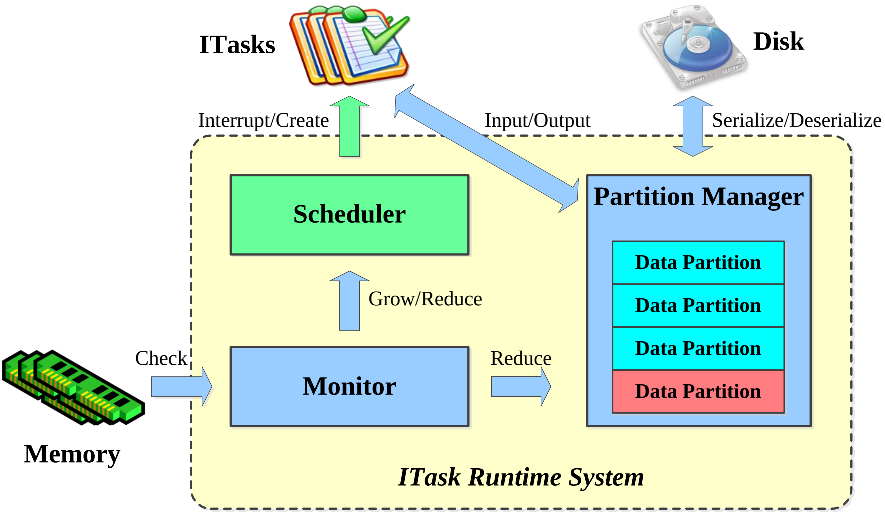
ITask Runtime System



ITask Runtime System



ITask Runtime System



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

Goal

- Show the effectiveness on real-world problems

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

Name	**Dataset**
Map-Side Aggregation (MSA)	Stack Overflow Full Dump
In-Map Combiner (IMC)	Wikipedia Full Dump
Inverted-Index Building (IIB)	Wikipedia Full Dump
Word Cooccurrence Matrix (WCM)	Wikipedia Full Dump
Customer Review Processing (CRP)	Wikipedia Sample Dump

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

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

Goal

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

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

Name	Dataset
WordCount (WC)	Yahoo Web Map and Its Subgraphs
Heap Sort (HS)	Yahoo Web Map and Its Subgraphs
Inverted Index (II)	Yahoo Web Map and Its Subgraphs
Hash Join (HJ)	TPC-H Data
Group By (GR)	TPC-H Data

Tuning Configurations for Original Programs

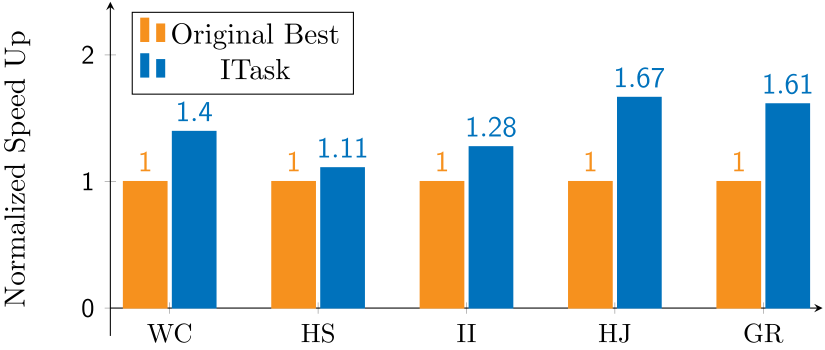
UCIrvine

Configurations for best performance

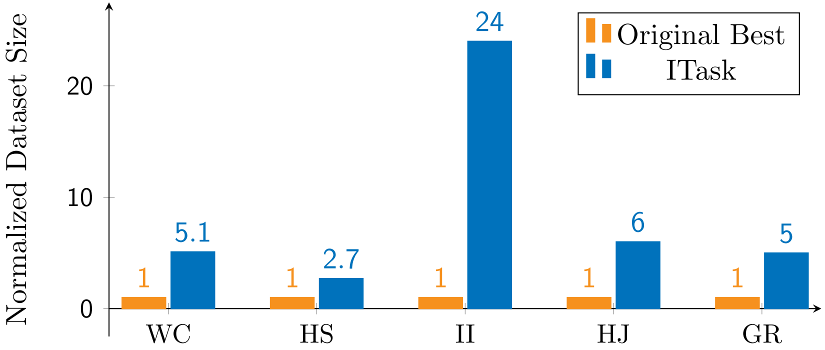
Name	Thread Number	Task Granularity
WordCount (WC)	2	32KB
Heap Sort (HS)	6	32KB
Inverted Index (II)	8	16KB
Hash Join (HJ)	8	32KB
Group By (GR)	6	16KB

Configurations for best scalability

Name	Thread Number	Task Granularity
WordCount (WC)	1	4KB
Heap Sort (HS)	1	4KB
Inverted Index (II)	1	4KB
Hash Join (HJ)	1	4KB
Group By (GR)	1	4KB



On average, ITask is 34.4% faster



On average, ITask scales to 6.3×+ larger datasets

ITask is pratical

- 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× larger datasets

A programming model + a runtime system

- Non-intrusive
- Easy to use

First general technique to amplify problems

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

A general performance testing framework

- Includes a compiler and a runtime system
- Very pratical

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

Publications

- 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