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]

[1]University of California, Irvine

[2]University 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 Spark HIVE mahout

Motivation

Data-parallel system

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

hadoop Spark HIVE mahout

⚠ Memory pressure on single nodes

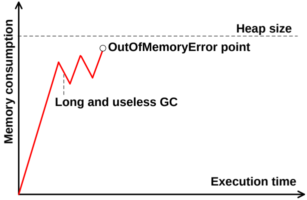
Our study

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

Memory Pressure in the Real World

Memory pressure on individual nodes

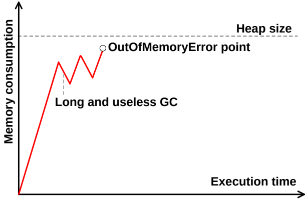
- Executions push heap limit (using managed language)
- Data-parallel systems struggle for memory



Memory Pressure in the Real World

Memory pressure on individual nodes

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

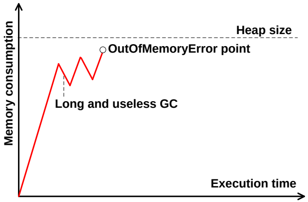


CRASH OutOfMemory Error

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

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

Root Cause 2: Large Intermediate Results

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

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

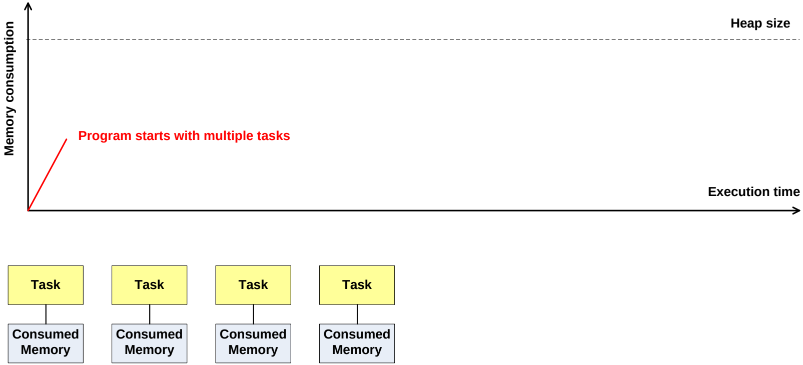
We need a systematic and effective solution!

Our Solution

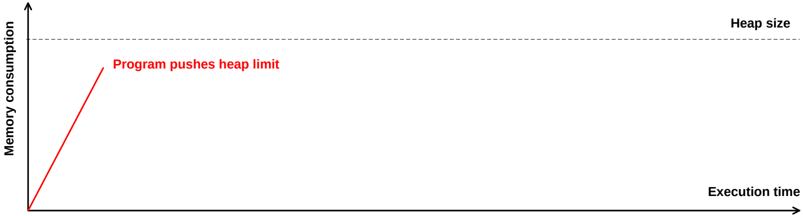
Interruptible **Task**: *treat memory pressure as interrupt*

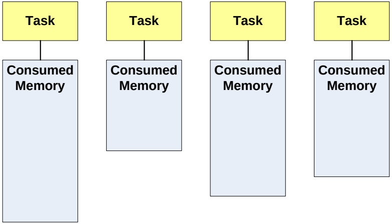
Dynamically change parallelism degree

Why Does Our Technique Help

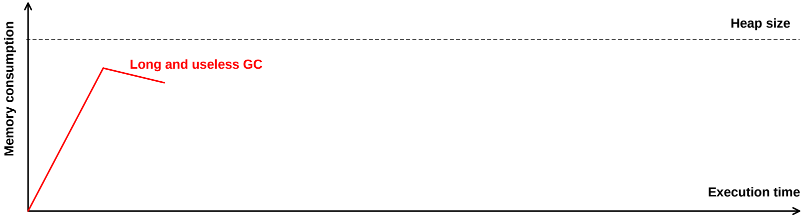


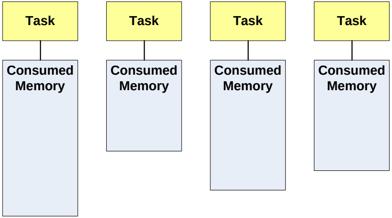
Why Does Our Technique Help



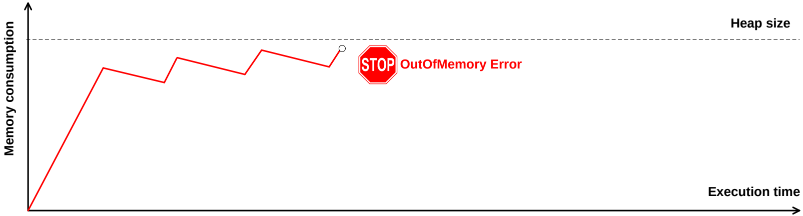


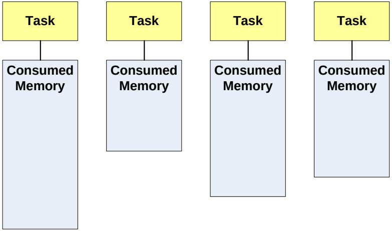
Why Does Our Technique Help



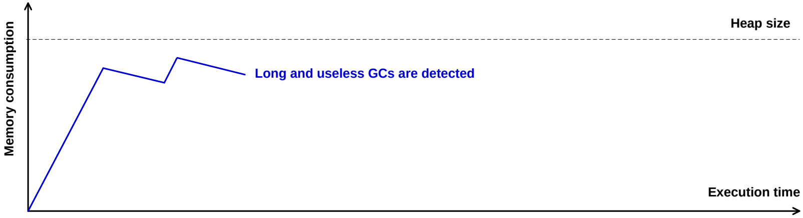


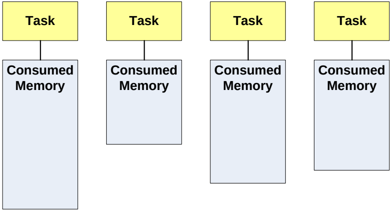
Why Does Our Technique Help



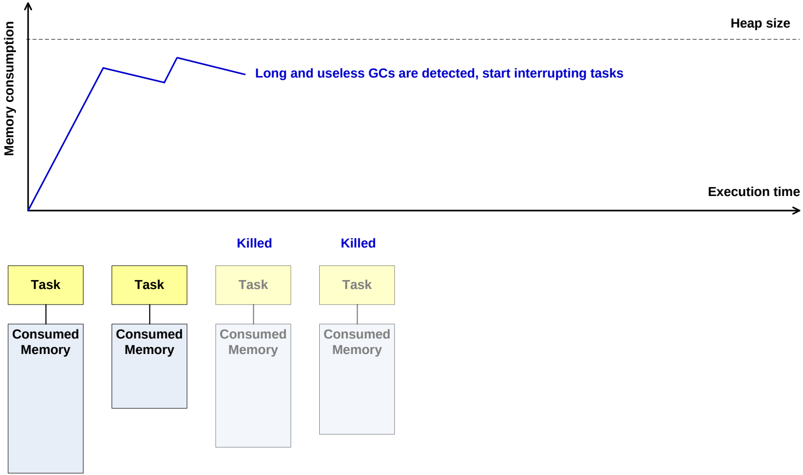


Why Does Our Technique Help

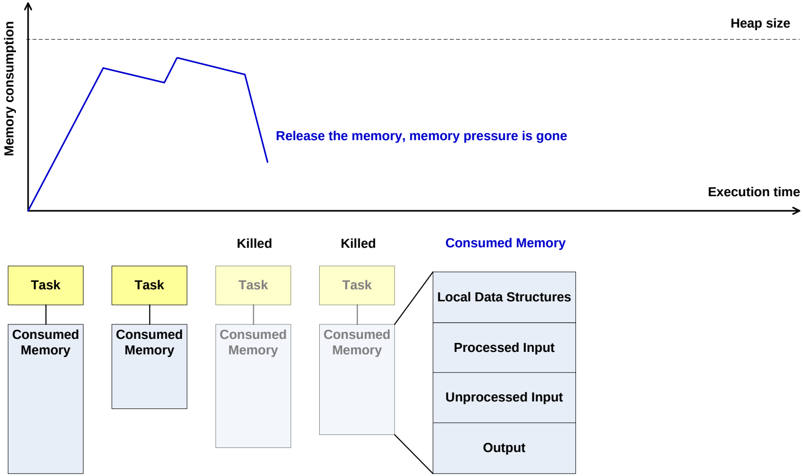




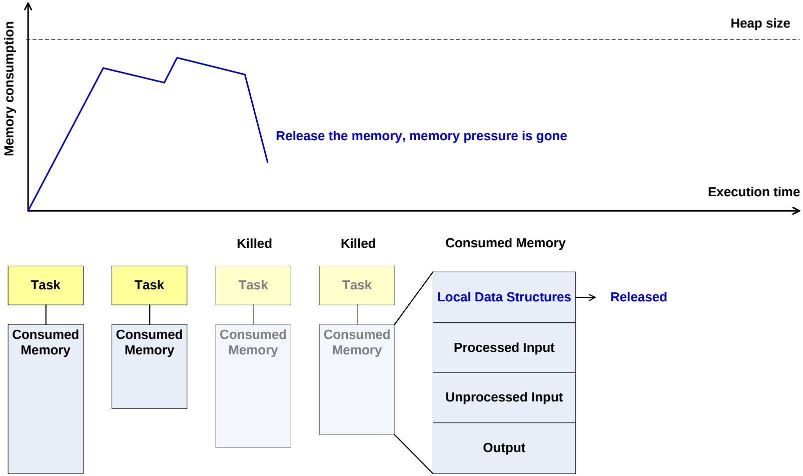
Why Does Our Technique Help



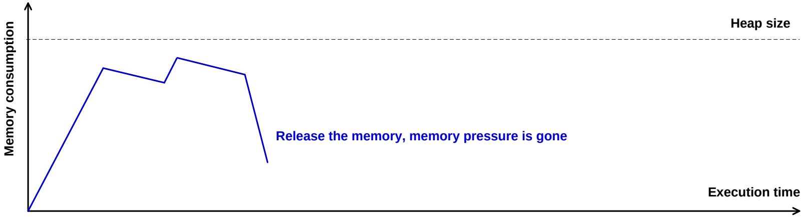
Why Does Our Technique Help



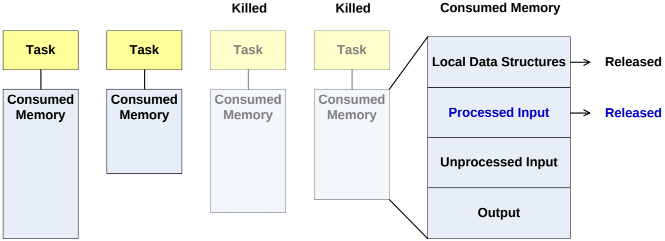
Why Does Our Technique Help



Why Does Our Technique Help



Release the memory, memory pressure is gone


Killed

Killed

Consumed Memory

Task

Consumed Memory

Task

Consumed Memory

Task

Consumed Memory

Task

Consumed Memory

Local Data Structures → Released

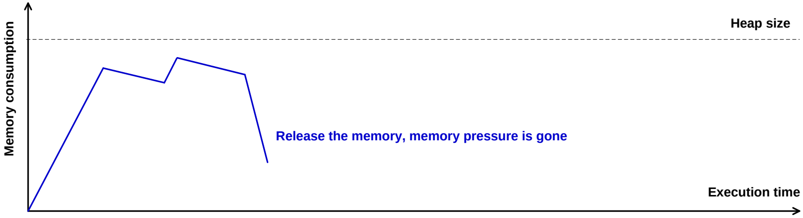
Processed Input → Released

Unprocessed Input

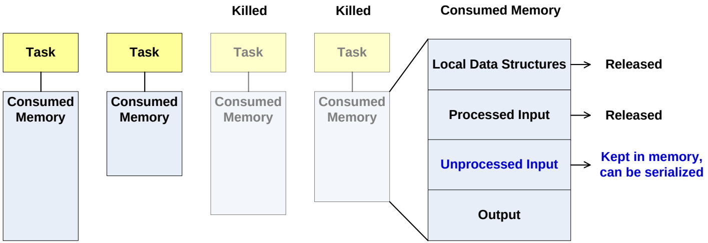
Output

Why Does Our Technique Help

UCIrvine

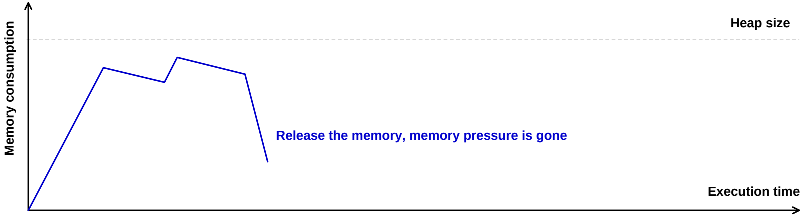


Release the memory, memory pressure is gone

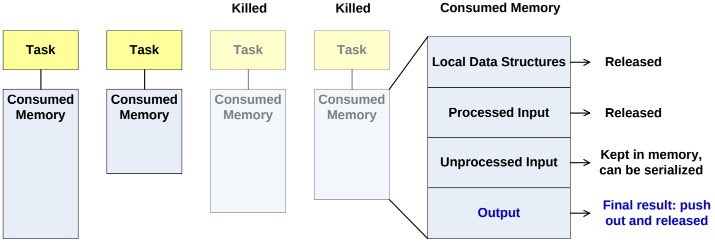


Kept in memory, can be serialized

Why Does Our Technique Help



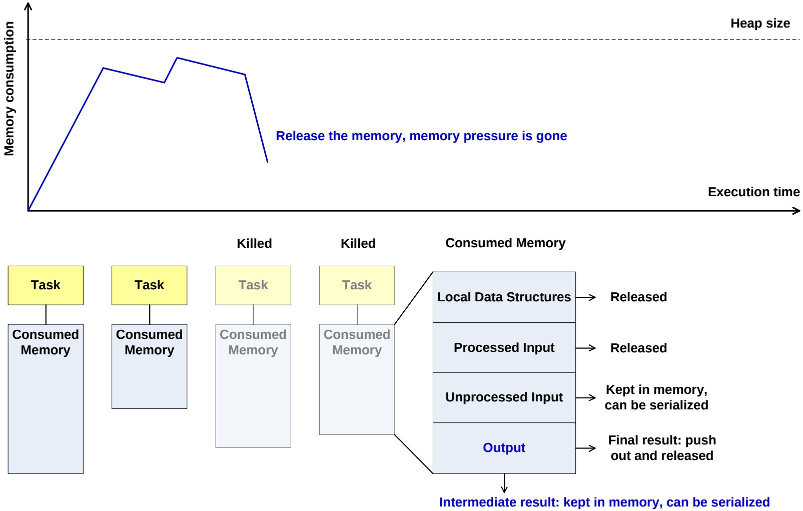
Release the memory, memory pressure is gone



Consumed Memory

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

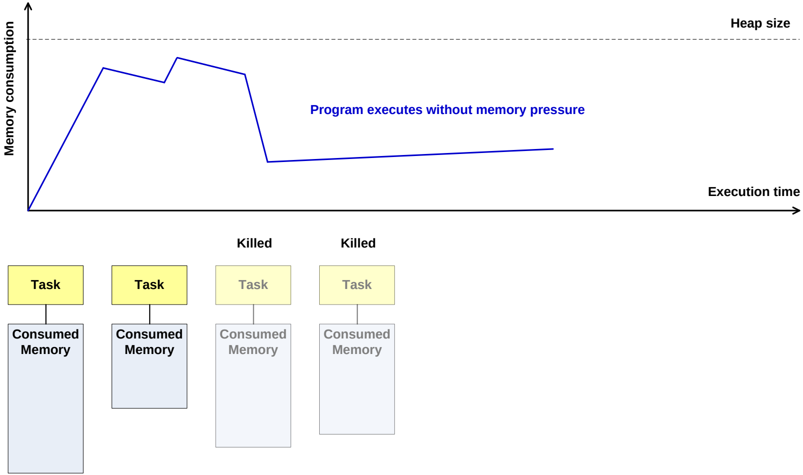
Why Does Our Technique Help



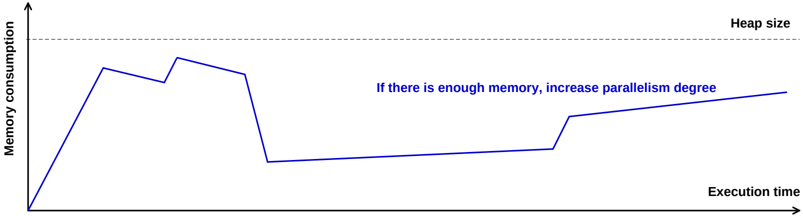
Release the memory, memory pressure is gone

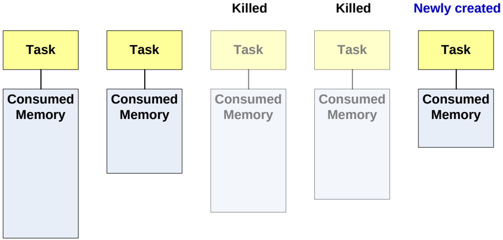
Intermediate result: kept in memory, can be serialized

Why Does Our Technique Help



Why Does Our Technique Help





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 → a programming model

How to interrupt/reactivate tasks → a runtime system

The Programming Model

UCIrvine

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

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

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
  ...
  // 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

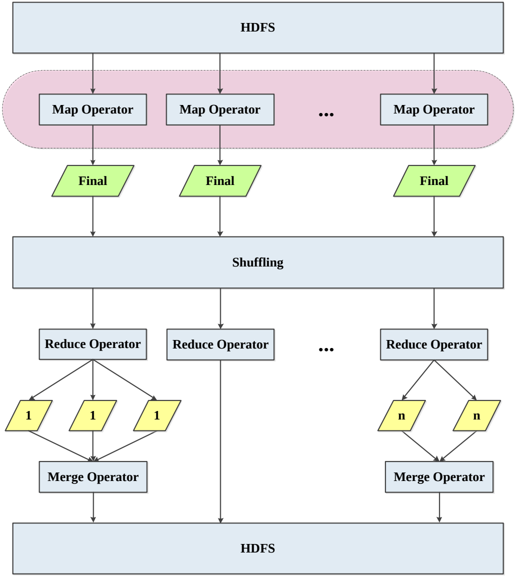
- How to merge intermediate results

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
  ...
  // 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;
  }
}
```

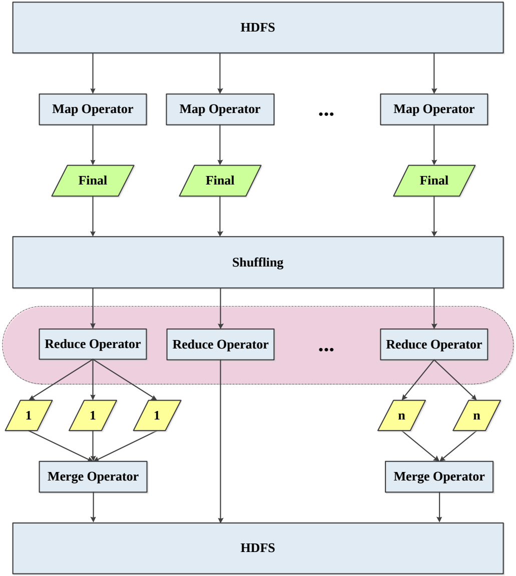
ITask WordCount on Hyracks



MapOperator

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

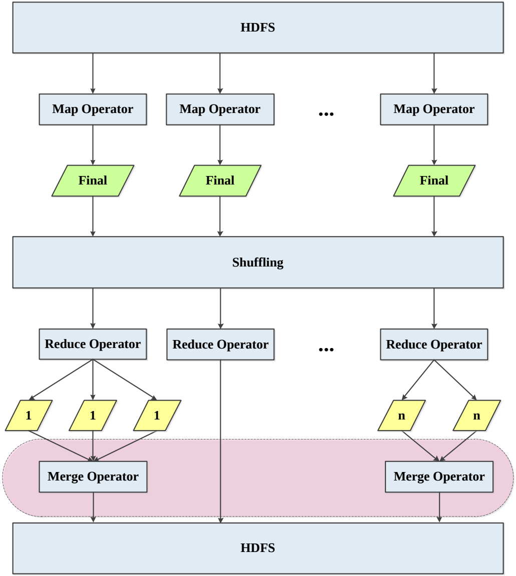
ITask WordCount on Hyracks



ReduceOperator

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

ITask WordCount on Hyracks



MergeOperator

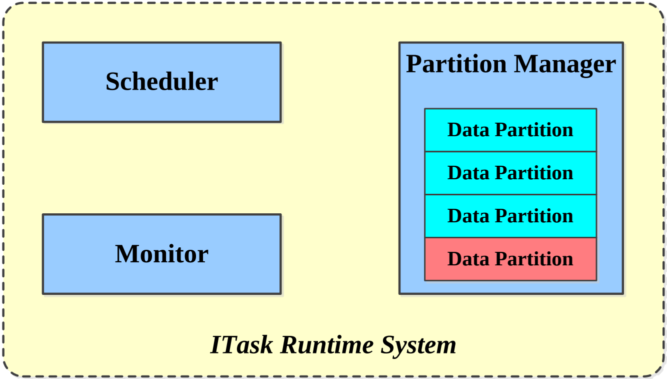
```
class MergeTask extends MITask {
  void interrupt() {
    // Tag the results;
    // Output as intermediate
    // results
  }
  // Some other fields and methods
  ...
}
```

Challenges

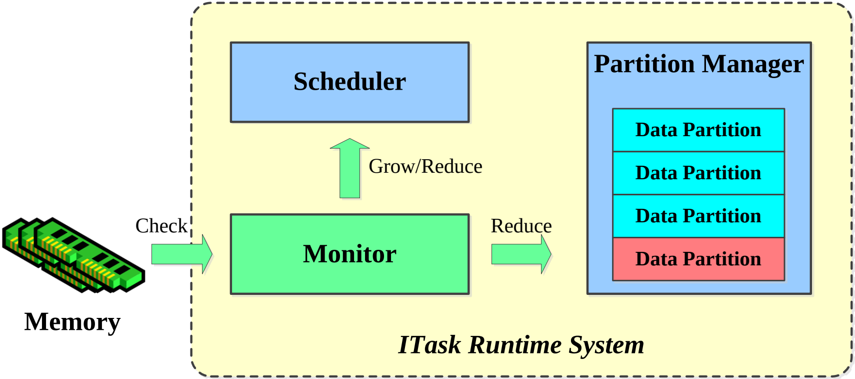
How to expose semantics → a programming model

How to interrupt/activate tasks → a runtime system

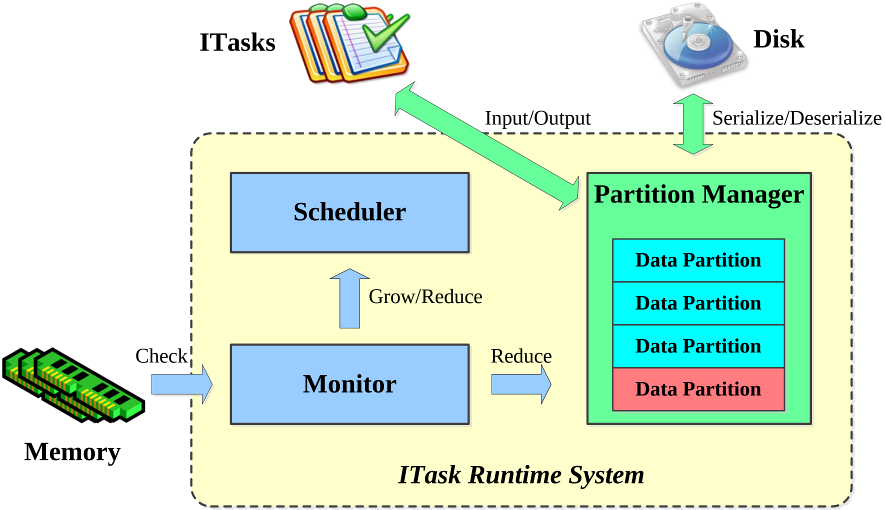
ITask Runtime System



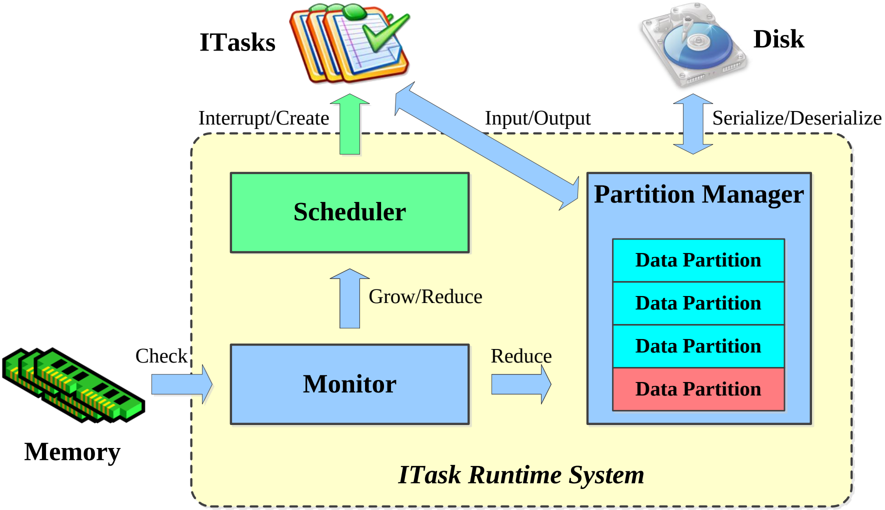
ITask Runtime System



ITask Runtime System



ITask Runtime System



Evaluation Environments

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

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

Improvements

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

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

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

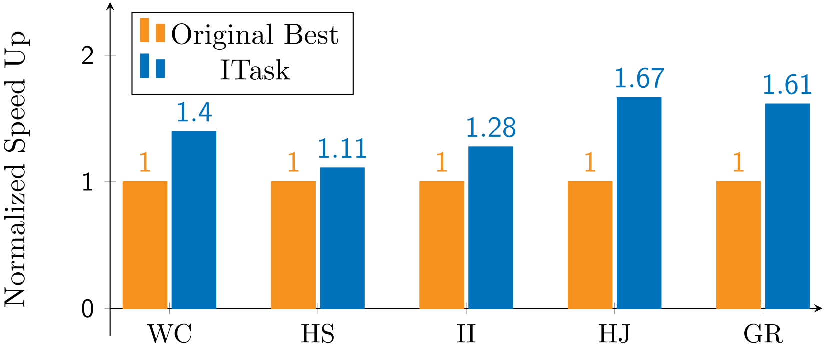
Configurations for best performance

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

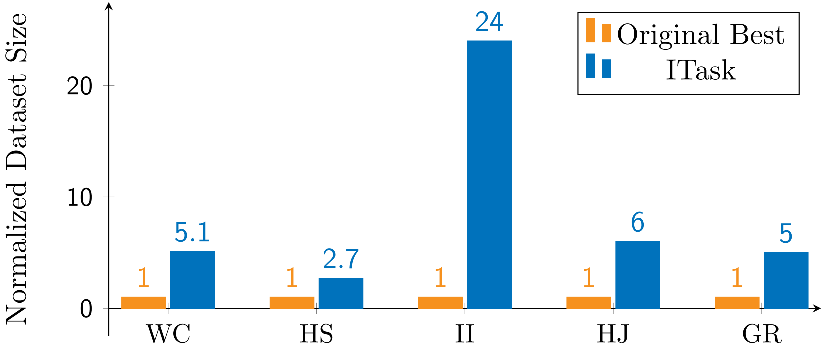
Name	Thread Number	Task Granularity
WordCount (WC)	1	4KB
Heap Sort (HS)	1	4KB
Inverted Index (II)	1	4KB
Hash Join (HJ)	1	4KB
Group By (GR)	1	4KB

Improvements on Performance



On average, ITask is 34.4% faster

Improvements on Scalability



On average, ITask scales to 6.3×+ 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