Interruptable Tasks: Treating Memory Pressure As Interrupts for Highly Scalable Data-Parallel Programs
Scalability issues remain common in Big Data systems

- Even on state-of-the-art frameworks, e.g., Hadoop, Spark, ...

An important problem: memory pressure on single node (Managed runtime system)

- OutOfMemory Error
- Long GC
Many factors may lead to memory pressure
- Inappropriate configurations
- Large intermediate results
- Third-party libraries
- ...

How to solve the problem?

Manual parameter tuning ≠ the ideal solution
Challenge: can we reduce the memory pressure
- Without additional hardware resource
- Without manual parameter tuning
- Without hurting the performance

Our Solution: **Interruptable Task**
- Treat memory pressure as interrupts
When an ITask is interrupted

Release the local variables

Release the processed portion of the input

Serialize the intermediate results to disk

Serialize the unprocessed portion of the input to disk

More available memory for the rest active tasks
The Design Choices

- Choice 1: hack the exiting system
- Choice 2: design a new language
- **Choice 3: library based implementation**
  
  Our choice, non-intrusive, easy to use
How to make existing tasks interruptable?

Extend the ITask abstract class

Four abstract methods

// The ITask abstract class in the library
abstract class ITask {
    abstract void initialize(); /* Initialization logic */
    abstract void interrupt(); /* Interrupt logic */
    abstract void cleanup(); /* Finalization logic */
    abstract void process(Tuple t); /* Process a tuple */

    /* Scalable loop */
    boolean scaleLoop(DataPartition dp) {
        while (dp.hasNext()) {
            if (Monitor.hasMemoryPressure() && ITaskScheduler.terminate(this)) {
                /* Invoke the user-defined interrupt logic */
                interrupt();
                /* Push the partially processed input to the queue*/
                ITaskScheduler.pushToQueue(dp);
                return false;
            }
            process(dp.next());
        }
        return true;
    }
}
The ITask Programming Model

- The input and output of an ITask
  - DataPartition objects

```java
// The DataPartition abstract class in the library
abstract class DataPartition {
  /* The tag for grouping */
  int tag;
  /* The cursor points to the first unprocessed tuple */
  int cursor;
  /* Return whether there exists unprocessed tuple */
  abstract boolean hasNext();
  /* Serialize the DataPartition */
  abstract void serialize();
  /* Deserialize the DataPartition */
  abstract DataPartition deserialize();
}
```
The Execution of An ITask

- Unprocessed Input DataPartition
- initialize
- Tuples
- process
- Tuple
- Memory Pressure
- interrupt
- Paritial Output DataPartition
- cleanup
- Output DataPartition
- scaleLoop
The ITask Runtime System

- **Monitor**
  - Monitor the resource condition

- **Partition Manager**
  - Serialize/Deserialize the data partitions between memory and disk

- **Scheduler**
  - Increase/Reduce ITask instances
Implementation and Evaluation

- ITask library implementation
  - Available on Hyracks 0.2.14 (newest version)
- Benchmarks
  - WordCount (WC), Heap Sort (HS), Inverted Index (II), Hash Join (HJ), and Group By (GR)
- Datasets
  - Yahoo Web Map for WC, HS and II
  - TPC-H data for HJ and GR
The execution time is reduced 39.54%. (1.65x faster)
The peak memory consumption is reduced 9.26%.
The ITask programs can scale up to 24.00x larger datasets
Thanks!
The First Main Novelty of ITask

- ITask works **proactively** in response to memory pressure
  - Take actions when an early bellwether of memory pressure is seen
  - Take the system back to the memory usage “safe zone” even before much time is spent on garbage collection (GC)
- Improve both scalability and performance
The Second Main Novelty of ITask

- ITask uses **a staged approach** to lower its memory consumption
  - Stage 1: releasing local variables
  - Stage 2: releasing the processed portion of the input
  - Stage 3: partial output
  - Stage 4: intermediate results
  - Stage 5: in-memory data, e.g., the rest of the unprocessed data in memory
- ITask lazily performs the releasing of the memory to minimize the negative impact
The Third Main Novelty of ITask

- ITask is non-intrusive and easy to implement
  - ITask programming model
    - Users only need to reconstruct their existing code and implement the abstract methods defined in ITask class.
  - ITask runtime system (IRS)
    - IRS sits on top of existing frameworks, provides complementary optimizations and additional safely guarantee.
    - IRS is a non-intrusive approach, and it’s easy to be integrated into existing frameworks.
ITask is the first attempt
1) to help data-parallel tasks survive memory pressure
2) and successfully scale to much larger datasets.
It also relieves the system from high GC costs resulting from frequent useless and long GCs.
ITask is a non-intrusive approach, and easy to use.