FMEM: A Fine-grained Memory Estimator for MapReduce Jobs

Lijie Xu[1,2], Jie Liu[1], and Jun Wei[1]

[1]Institute of Software, Chinese Academy of Sciences

[2]University of Chinese Academy of Sciences

6/26/2013

Outline

- What is FMEM?
- Why do we develop it?
- Is it a hard problem?
- How to develop it?
- How to estimate the fine-grained memory usage?
- Evaluation
- Related work
- Conclusion

What is FMEM?

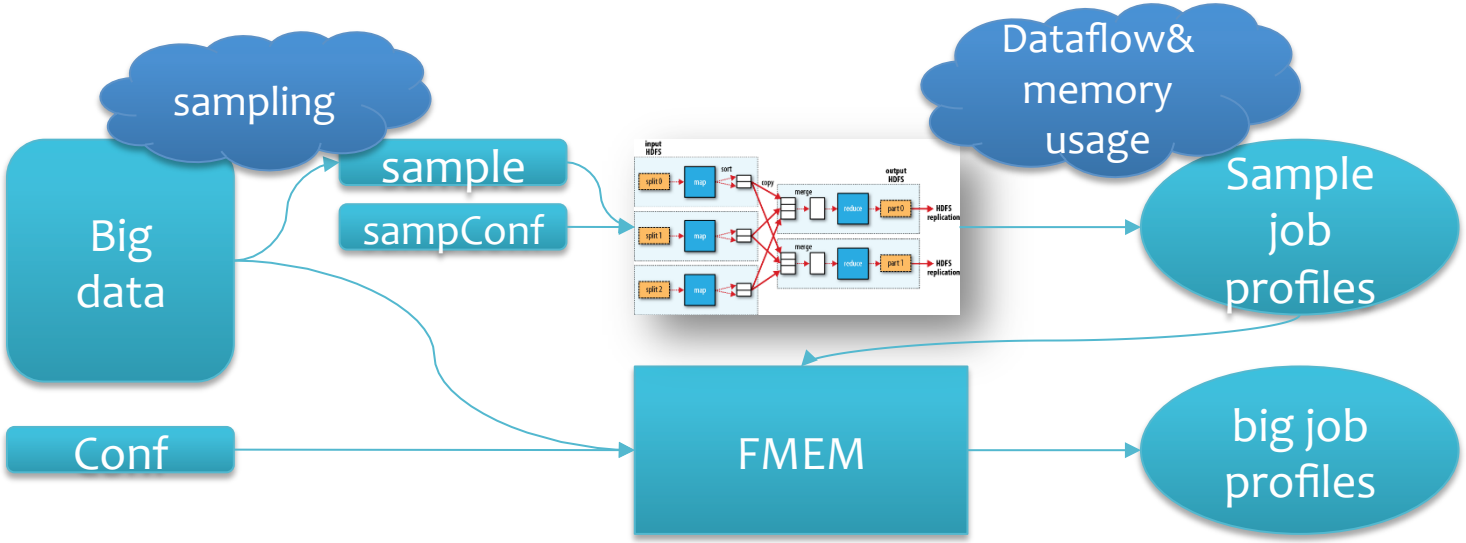
- It is designed to **analyze, predict** and optimize the fine-grained memory usage of map and reduce tasks.

Why do we want to develop FMEM?

- For users
  - feel hard to specify the memory-related configurations.
    - e.g., buffer size, reducer number, Xmx, Xms, and etc.
  - OutOfMemory error or performance degradation
- For task schedulers
  - e.g., YARN and Mesos
  - they allocate resource and schedule tasks according to the CPU and memory requirement of mappers/ reducers.
- For Hadoop committers
  - design a more memory-efficient framework.

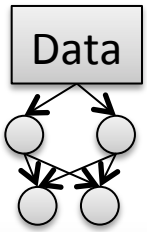
So, what is the fundamental problem?

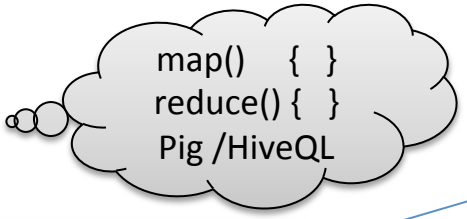
- Tell job's memory usage before running it on big dataset.
- *f* (Data *d*, Conf *f*, Program *p*, Resource *r*)
  → memory usage *mu* of mappers and reducers
- Add <SampData *sd*, SampConf *sf*, SampProfiles *sp*>
  → memory usage *mu* of real mappers and reducers



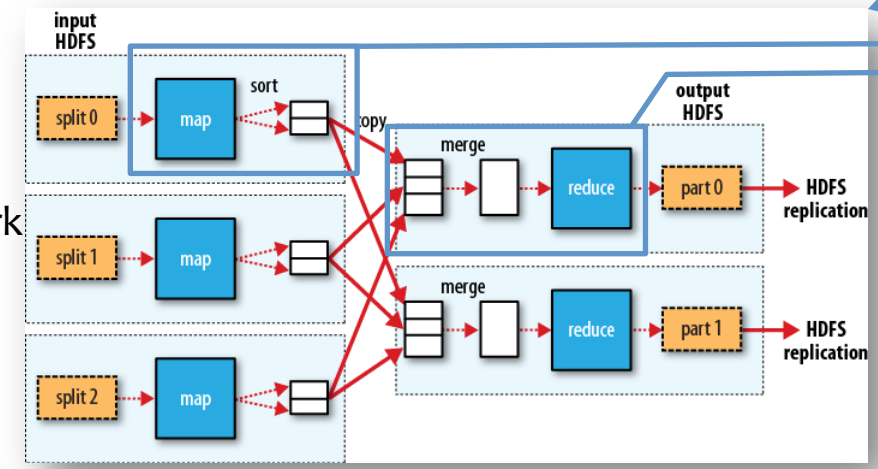
Is it a hard problem?

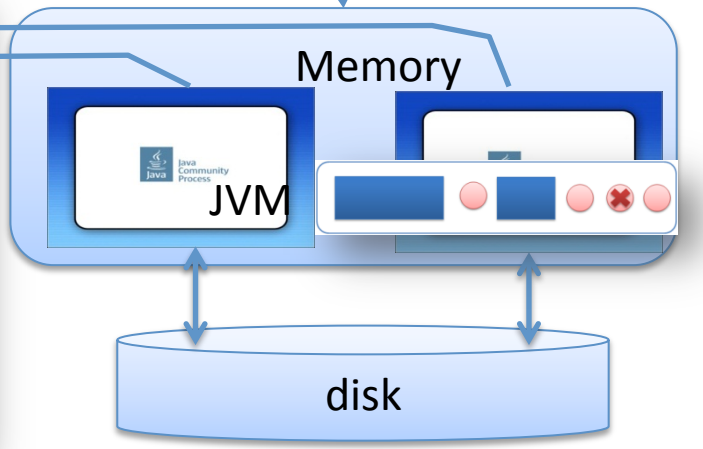
User layer



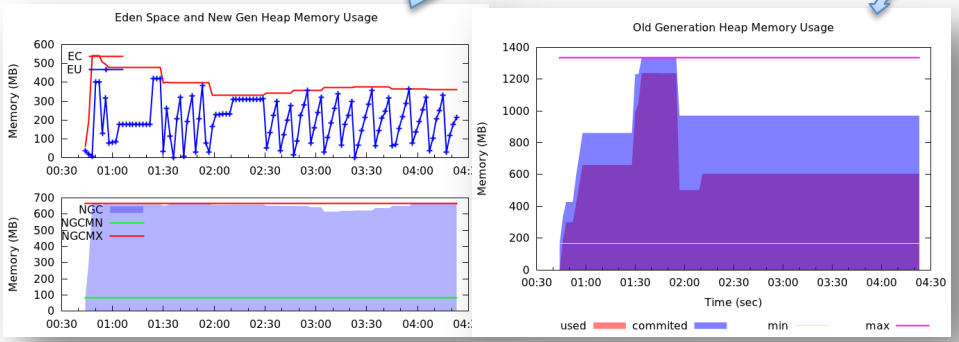


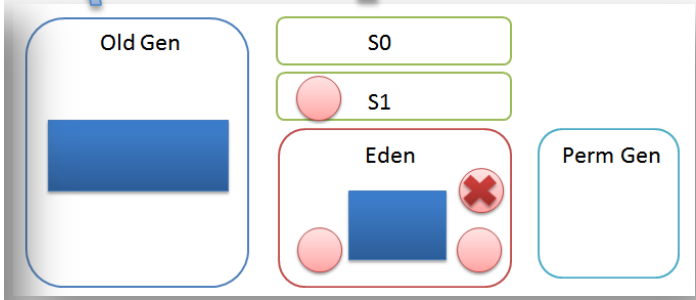
Framework layer





Execution layer





How to do it? dataflow→objects→usage

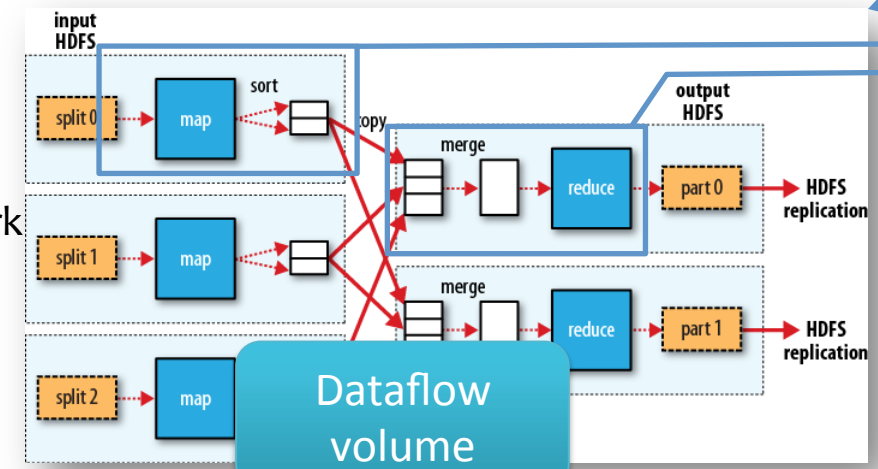
User layer

Data

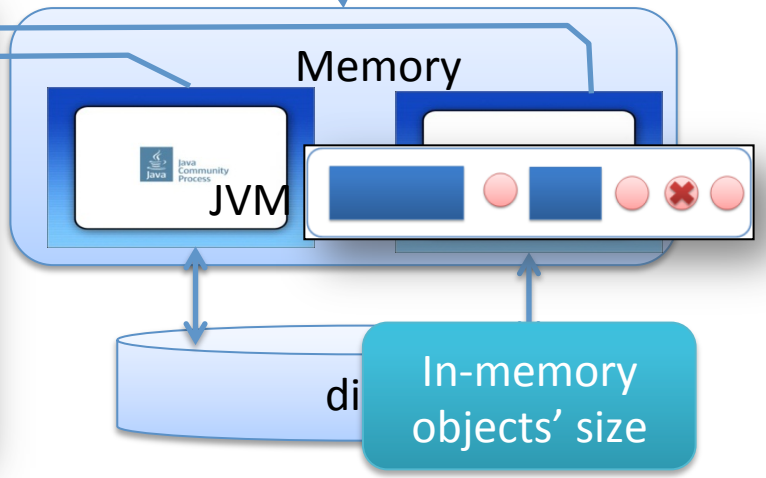
map() { }
reduce() { }
Pig /HiveQL

Configuration

Framework layer

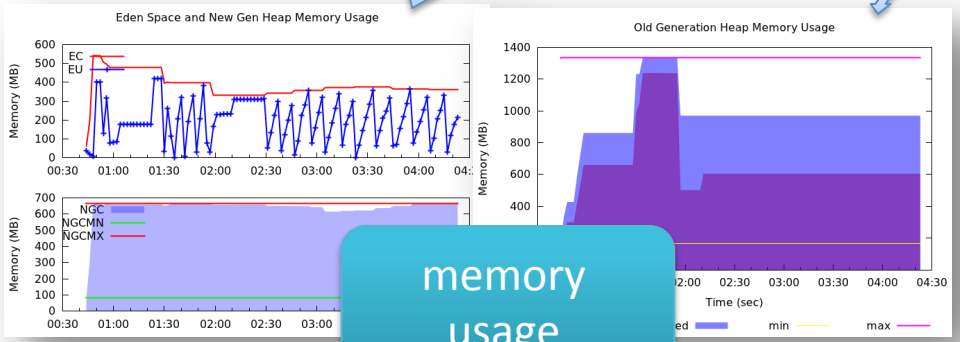


Dataflow volume

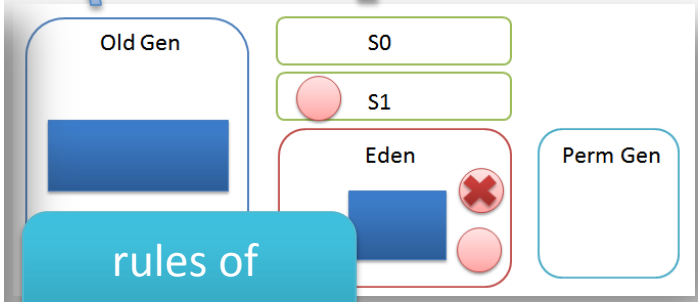


In-memory objects' size

Execution layer



memory usage

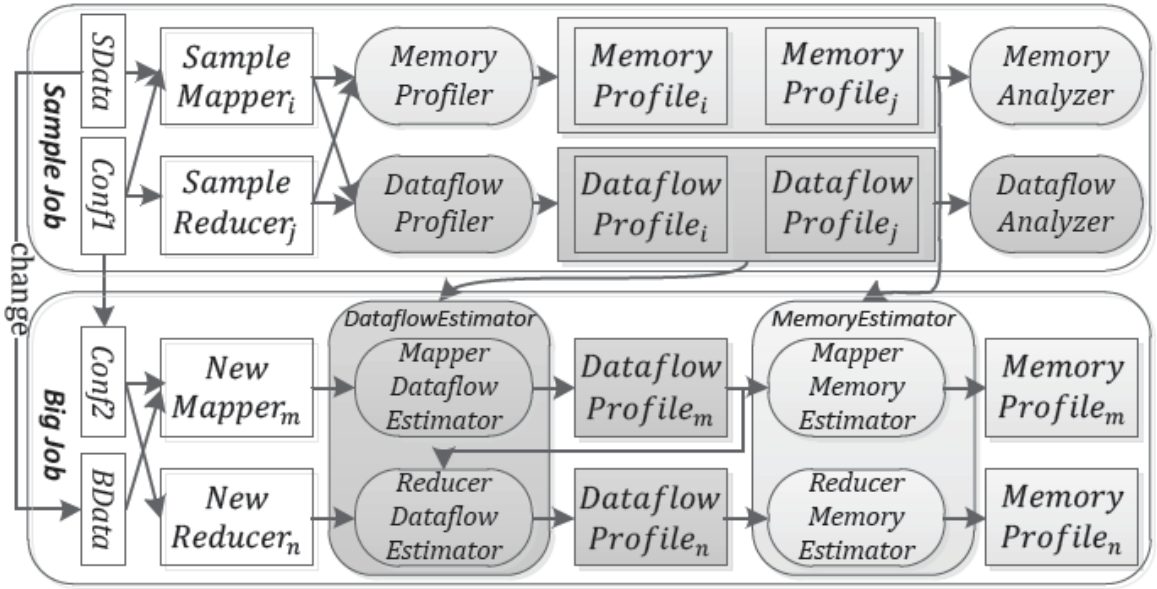


rules of memory usage

How to do it? dataflow→objects→usage

- Built-in Monitor
  - get Records/Bytes Statistics & real-time memory usage
- Profiler
  - count intermediate data and TempObjs in each phase

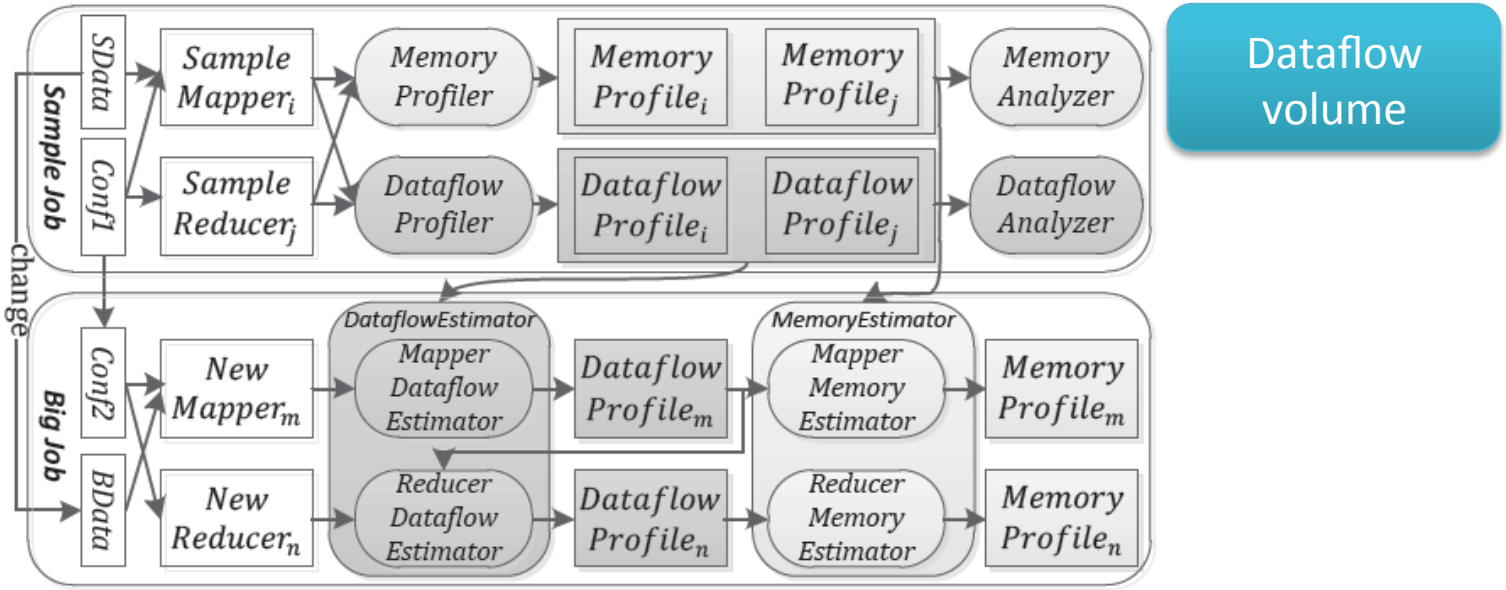
Instrument get statistics



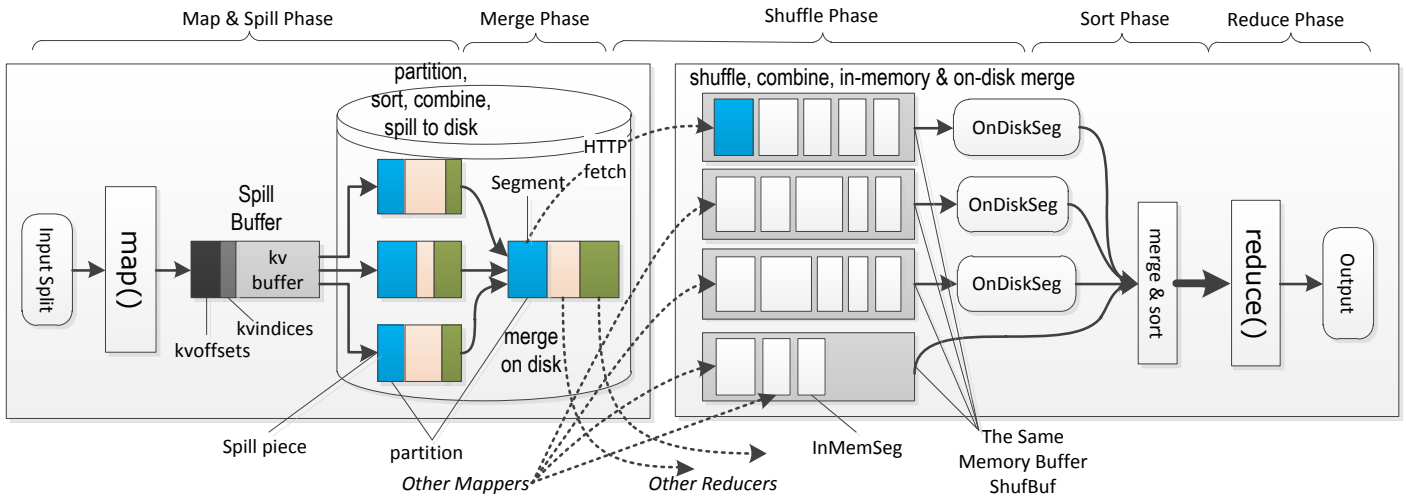
Dataflow volume

How to do it? dataflow→objects→usage

- Estimate the dataflow
  - We actually build a MapReduce simulator to estimate the intermediate data under specific <Data *d*, Conf *f*, sample profiles *sp*>.



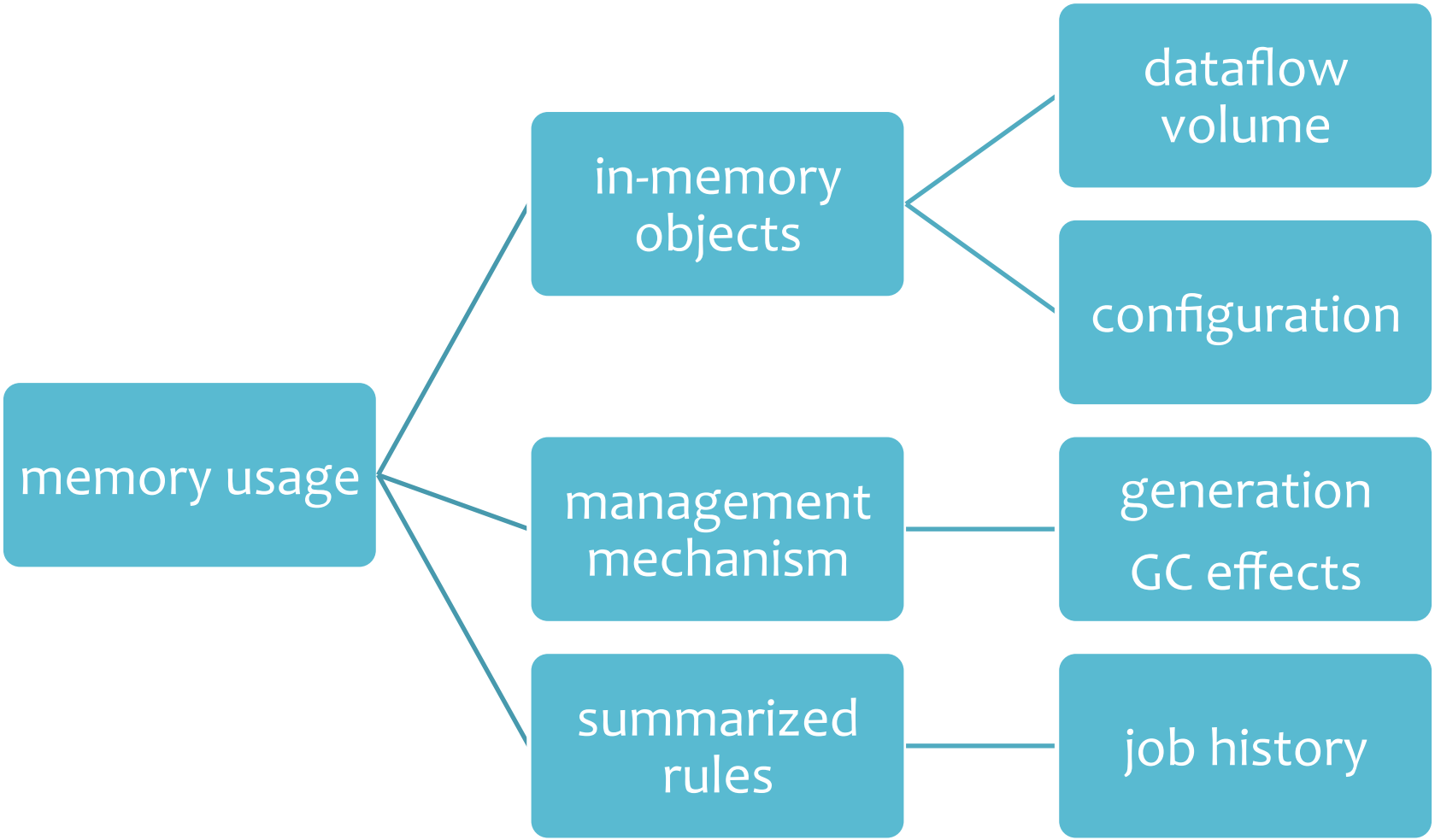
Which are memory-consuming objects?



- **Memory buffer** (configuration)
  - e.g., spill buffer (*io.sort.mb*) and user-defined arrays
- **<K, V> records** (dataflow)
  - map() outputs records into spill buffer, in-memory segments in shuffle buffer
- **Temporary objects** (dataflow and programs)
  - byproducts of records such as char[], byte[], String and ArrayList
  - A WordCount mapper produces 480MB *java.nio.HeapCharBuffer* objects with 64MB input split.
- **Others**
  - code segment, native libraries, call stack and etc.

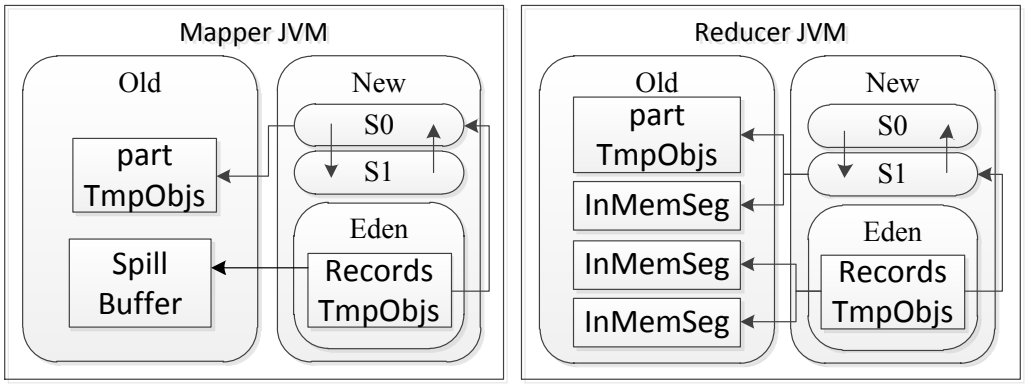
In-memory objects' size

How to do it? dataflow→objects→usage



How to do it? dataflow→objects→usage

- Rules-statistics based approach



Size of in-memory objects

Management mechanism

GC effects

- RULE 1. $MapOU \approx f(conf)$
- RULE 2. $MapNGU \approx f(Xmx, Records, TempObjs)$
- RULE 3. $RedNGU \approx f(Xms, Xmx, Records, TempObjs)$
- $RedOU \approx f(Xms, Xmx, Records, TempObjs)$

rules of memory usage

How to do it? dataflow→objects→usage

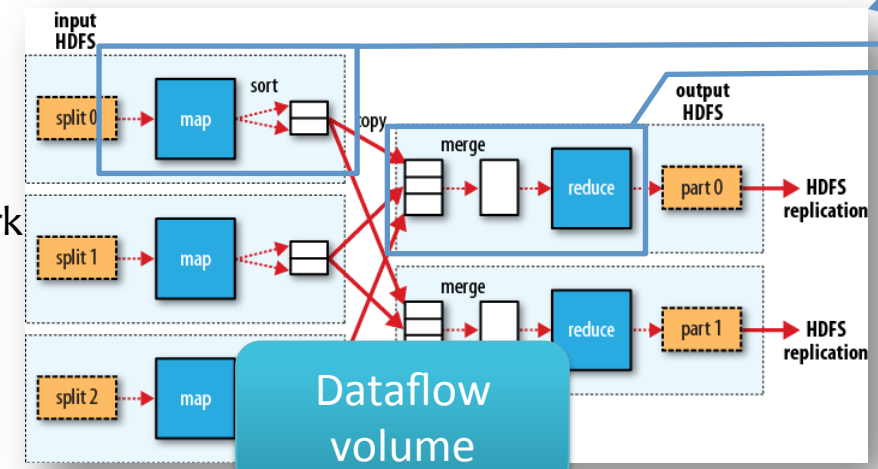
User layer

Data

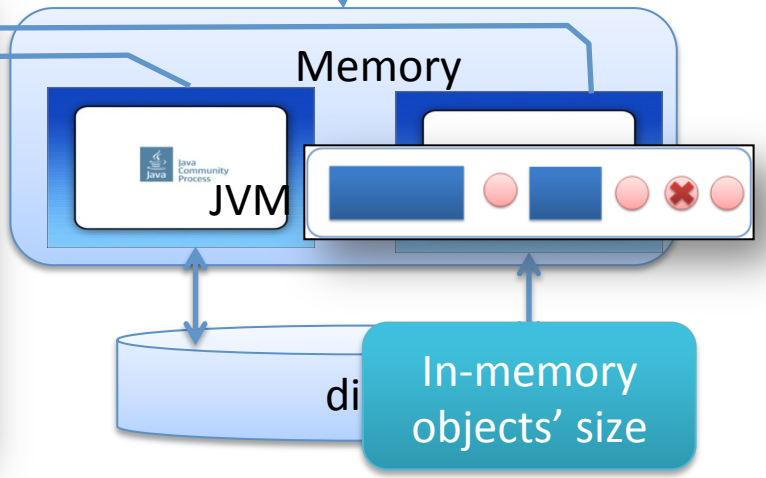
map() { }
reduce() { }
Pig /HiveQL

Configuration

Framework layer

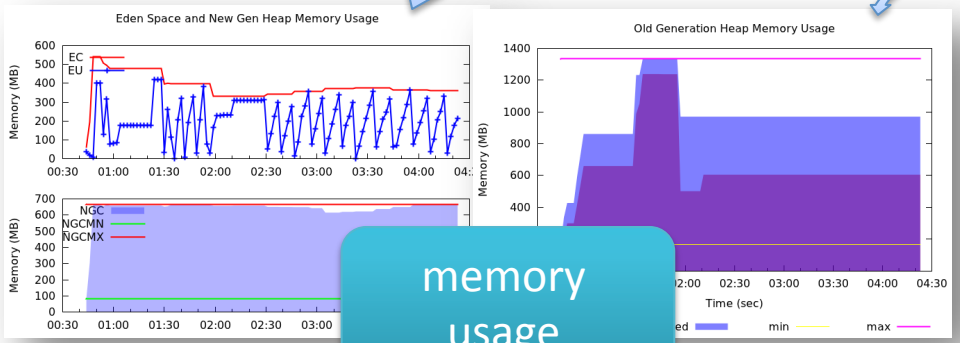


Dataflow volume

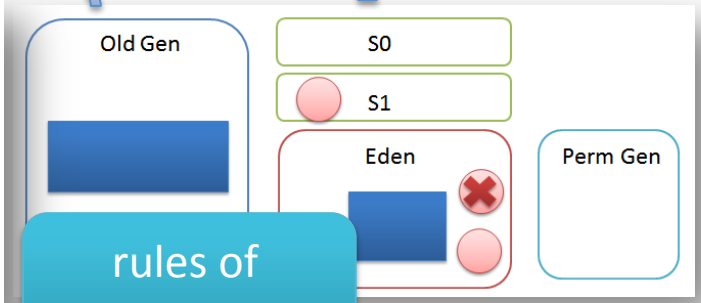


In-memory objects' size

Execution layer



memory usage



rules of memory usage

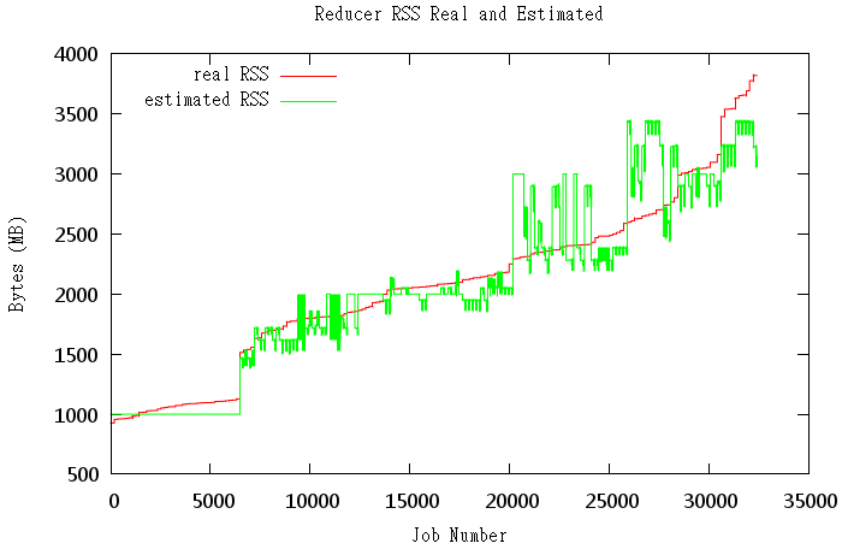
Evaluation

Table 1: Representative MapReduce Applications

Applications	Dataset	Combine	Compress
WikiWordCount	9.4 GB	Y	N
BuildInvertedIndex	9.4 GB	N	SeqBlock
UserVisits_Aggre-pig	75 GB	Y	N
TwitterBiEdgeCount	24.4 GB	N	N
TeraSort	36 GB	N	Y

Buffer size	Reducer number
Max-min heap size	Split size

$$relative\ error = |emu - rmu/rmu| * 100\%$$



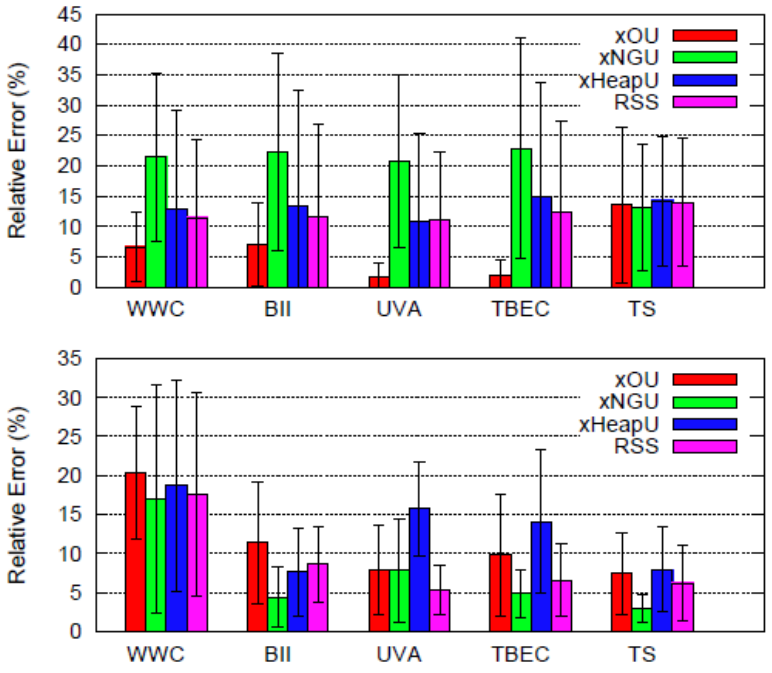


Figure 2: Relative error with standard deviation of <*emu*,*rmu*>

How about the related work?

- Execution time estimation
  - ParaTimer, KAMD, ARIA and etc.
- Find optimum configurations
  - Starfish can predict job's performance (mainly runtime) with different configurations and has a cost-based optimizer to find the optimum configuration.
- Find optimum GC policy for multicore MapReduce
  - Garbage collection auto-tuning for java MapReduce on multi-cores, ISMM 2011

Conclusion and discussion

- Provide a detailed analysis of job's memory usage
  - considering dataflow and memory management from user-level to JVM internals.
- Develop a fine-grained memory estimator
  - predict job's memory usage in a large space of configurations.
- FMEM will be improved to do auto-configuration, auto-optimization and etc.
  - JIRA MAPREDUCE-4882 & MAPREDUCE-4883

Thanks, Q&A

ISCAS
中国科学院软件研究所
Institute of Software Chinese Academy of Sciences