Facade: A Compiler and Runtime for (Almost) Object-Bounded Big Data Applications
Khánh Nguyen, Kai Wang, Yingyi Bu, Lu Fang, Jianfei Hu, Harry Xu
University of California, Irvine

Motivation
Design a scalable and efficient system is a key challenge to both researchers and practitioners
- Mainstream approach is to enable parallelism by using a large number of machines
- Typical parallel frameworks such as MapReduce, GraphHive, or Pig use Java, a managed language which comes with a managed runtime system
- When object-orientation meets Big Data, the cost of managing runtime system becomes the bottleneck
- Significantly reduced scalability, JVM crashes even if the size of the processing dataset is much smaller than the heap size
- Poor performance is inherent with the managed runtime system and remains a serious problem despite many optimizations from various research communities

Related Work
- Optimizations of Big Data applications:
  - Data pipeline: [Agrawal VLDB’08], Flume-Java [Chambers PVLDB’13, Dyad,Java] [Yu OSDI’08]
  - MapReduce-related: Hive [Husso ICDE’10], Panacea [Lu CGO’13]
- Techniques for reducing runtime management costs
  - [Jain PVLDB’13], [Hallenberg PVLDB’13], [Hix ISMM’04]
  - Immix [Blackburn PVLDB’13], [32]
  - Prolific types [Shuf POPL’02]
- Techniques for reducing numbers of objects
  - Object pooling and certain design patterns
  - Object inlining:
  - [Dolby PLDI’00]
  - Prolific types
  - [Aiken PLDI’95], [Hallenberg PLDI’02], [Hick ISMM’04]
- MapReduce-related: Hive [Thusoo ICDE’10], Panacea [Liu CGO’13]
- Data pipeline:
  - [Agrawal VLDB’08], Flume-Java [Chambers PVLDB’13, Dyad,Java] [Yu OSDI’08]
- A high-performance graph analytical framework that enables efficient processing of large graphs on a single machine
- Normalized performance of the Facade-generated programs:
  - Word Count and the original
  - Page Rank
  - KMeans
  - Random Walk

Implementation and Evaluation
- Implemented using the Soot framework [Vallée-Rai CC’00]
- Most of the Java 7 features are supported
- 40+ KLOC; 1.5 years of development
- Fast translation speed: 550 instructions per second on average

GraphChi [Kryola OSDI’12]
A high-performance graph analytical framework that enables efficient processing of large graphs on a single machine
- Normalized performance of the Facade-generated programs:
  - Page Rank
  - KMeans
  - Random Walk

CPU Utilization Comparison

Hyracks [Borkar ICDE’11]
A data parallel platform to run data-intensive jobs on a cluster of shared-nothing machines
- Normalized performance of Facade-generated External Sort

GPS [Salihoglu SDBM’13]
A distributed graph processing system developed for scalable processing of large graphs
- Improvements on Page Rank, KMeans and Random Walk:
  - 3-15% running time reduction
  - 3-15% reduction in memory usage
  - 10% reduction in running time (avg.) max 48%
  - 1.4x improvement in throughput

Conclusions
A complete, non-intrusive package with a compiler that can automatically transform existing programs and a runtime system that runs on top of a JVM
- Experimental results show significant improvement in execution time, memory consumption, and scalability

Golden Rule for Scalability
- The number of heap objects and references must not grow proportionally with the cardinality of the dataset
- Formally, Facade guarantees a static bound:
  \[ O(s) \]
  - \( s \): cardinality of the dataset
  - \( O(n^m + n^p) \)
    - \( n \): number of data types
    - \( p \): number of pages
- Although it and \( n \) cannot be bounded statically, they are usually very small, hence the total number of objects is “almost” statically bounded
- The reduction is in many orders of magnitudes: in PageRank (GraphChi) 14 billions data objects are reduced to 1363 objects

Iteration-based Memory Management
- Iteration definition
  - Iterations are easy to identify; Facade relies on a user-provided pair of calls to start/finish iterations
  - Nested iterations are supported
- Allocation
  - Customized, high-performance allocator
  - Create a page manager per thread and per iteration to control memory: page managers form a hierarchy
  - Each page manager is associated with a pair \(<iterationID, thread>\)
- Contiguity: Continuous spaces of memory at data locality
- Support mostly thread-local data allocation
- Deallocation:
  - Once an iteration ends, reclamation can be safely done concurrently on the sub-tree rooted at the current page manager

Concurrency Support
- Thread-local pooling for facade objects
  - Each shared pool looks like a special lock class
  - The number of locks simultaneously needed is bounded by the number of threads