Pufferfish: Container-driven Elastic Memory Management for Data-intensive Applications

Wei Chen, Aidi Pi, Shaoqi Wang and Xiaobo Zhou
University of Colorado, Colorado Springs
Outline

• Introduction to data-intensive applications
• Memory problems and opportunities
• Pufferfish mechanisms
• Pufferfish architecture
• Evaluation
• Conclusion
Data-intensive applications

- Data analytics applications are extensively used in both industry and academia
- Most of the frameworks run on JVM
Data-intensive applications in clusters

- Executor memory is bounded by JVM heap size
- All executors of the same application share the same configuration
- Memory adjustment cannot be done at runtime
State-of-the-art

• JVM heap management
  • Analysis of data-intensive application behaviors
  • Improved garbage collection
  • ROLP[Eurosys’19], FACADE[SOSP’15], Yak[OSDI’16]
• Memory elasticity
  • Dynamically adjust memory allocation at runtime
  • C. Iorgulescu et al. [ATC’17], J. Wang et al. [ATC’17]
• Memory ballooning for virtual machines
  • Memory elasticity of virtual machines
Memory problems in clusters

• Garbage collection degrades job performance
• Memory under-utilization
• Out of memory error
  • Mis-configuration
  • Data skew
  • Load imbalance
  • ...

[Image of error message: "Out of memory."]
Illustration of memory problems

- Expensive garbage collection degrades performance
- Heterogeneous memory usage across executors in an application
Opportunities

• Memory heterogeneity
  • Memory is provisioned for the largest executor of the workload
  • Memory underutilization for small executors

• Memory Dynamics
  • Memory usage is dynamic during execution of a executor
  • Transient idle memory can be exploited
Pufferfish mechanisms

• Configure executors with a large JVM heap size.
• Configure executors with a small Docker memory limit
• Container-based executor memory management
  • **Puff** (increase) container memory limit on demand
  • **Suspend** an *Out-of-Container-Memory* container
  • **Resume** a task when memory is available

• A large JVM heap size always presents sufficient memory to executors
• Executors under memory pressure are swapped into disks instead of Out-Of-Memory error
  • Preserve job progress
Executor suspension and resumption

- An *Out-of-Container-Memory* executor incurs extensive disk I/O due to swapping.
- Heuristic: Suspend the executor by throttling its CPU usage to 1% when it is out of its container memory.

Tasks under suspension are still alive
- I/O activities are throttled.
Pufferfish architecture

- Container monitor
  - Performs container suspend and resume operations on FLEX containers

- Memory manager
  - Decides how much memory should be allocated to each container

- Resource scheduler plugin
  - Enforce fairness when taking account of different types of workloads
FLEX container

- FLEX container: a type of flexible container
- FLEX containers are set with a large JVM heap size
- FLEX containers are started the same small container memory limit
- FLEX containers are allowed to puff when its memory demand is larger than the container memory limit
Container monitor: an example

- Both executor 1 and executor 2 are configured with 16GB JVM heap and 2 GB container memory limit
- Container memory grows from 2GB with the increase of executor memory demand
Container monitor: an example

- Container constrains the actual physical memory
- Executor 1 demands 8GB, suspended at 4GB.
- Executor 2 demands 12GB, fully satisfied.
Memory manager

- Address memory contention
- Backoff-based puff
  - Increase the container size according to their priorities
- Kill the container with the lowest priority when memory is used up
Pufferfish scheduling plugin

- Scheduling Plugin
  - Exposes physical memory usage of each node
  - Balances the physical memory usage across nodes

- Prioritization Policies
  - Earliest Job First (EJF): Puff the earliest submitted job first
  - Shortest Job First (SJF): Puff the shortest job first
Evaluation setup

• Setup
  • 26-node cluster with Ubuntu-16.04
  • 32 cores, 128GB RAM, RAID-5 HDDs
  • Cluster is connected by 10Gbps Ethernet
  • Hadoop-2.7.2, Spark-2.0.1, Docker-1.12.1

• Workloads
  • HiBench as batch workloads
  • TPC-H on Spark-SQL as latency-critical workloads
Single node

- Workloads: Kmeans and Wordcount
- Pufferfish vs. Yarn with different heap sizes
- Pufferfish achieves the best performance for **Kmeans**
  - **Kmeans** is dominated by GC and is CPU intensive
- Pufferfish achieves close-optimal performance for **Wordcount**
  - **Wordcount** is I/O intensive
  - Higher parallelism outweighs a larger heap size
Production trace

- Replay a subset of Google trace in the 26-node cluster
- Pufferfish completes all workloads without OOM
- Pufferfish achieves the highest memory utilization
Mixed workloads

- Workloads
  - 38 data-intensive jobs as batch jobs
  - 576 TPC-H jobs as latency-critical jobs
- For **latency-critical** workloads, Pufferfish achieves almost the same performance as stand-alone execution
- For **batch workloads**, Pufferfish outperforms default Yarn with 64GB heap by **adaptive parallelism**
Conclusion

• Data-intensive applications suffer from memory issues OOM and suboptimal memory utilization.

• Pufferfish is an elastic memory manager that leverage OS containers to achieve dynamical memory allocation: puff/suspend/reclaim

• Pufferfish can avoid OOM, preserve job performance and improve cluster memory utilization
Pufferfish: Container-driven Elastic Memory Management for Data-intensive Applications

Wei Chen, Aidi Pi, Shaoqi Wang and Xiaobo Zhou
University of Colorado, Colorado Springs

Thank you!

Q & A