MnemonicProposal

Mnemonic Proposal

Abstract

Mnemonic is a Java based non-volatile memory library for in-place structured data processing and computing. It is a solution for generic object and block persistence on heterogeneous block and byte-addressable devices, such as DRAM, persistent memory, NVMe, SSD, and cloud network storage.

Proposal

Mnemonic is a structured data persistence in-memory in-place library for Java-based applications and frameworks. It provides unified interfaces for data manipulation on heterogeneous block/byte-addressable devices, such as DRAM, persistent memory, NVMe, SSD, and cloud network devices.

The design motivation for this project is to create a non-volatile programming paradigm for in-memory data object persistence, in-memory data objects caching, and JNI-less IPC. Mnemonic simplifies the usage of data object caching, persistence, and JNI-less IPC for massive object oriented structural datasets.

Mnemonic defines Non-Volatile Java objects that store data fields in persistent memory and storage. During the program runtime, only methods and volatile fields are instantiated in Java heap, Non-Volatile data fields are directly accessed via GET/SET operation to and from persistent memory and storage. Mnemonic avoids SerDes and significantly reduces amount of garbage in Java heap.

Major features of Mnemonic:

- Provides an abstract level of viewpoint to utilize heterogeneous block/byte-addressable device as a whole (e.g., DRAM, persistent memory, NVMe, SSD, HD, cloud network Storage).
- Provides seamless support object oriented design and programming without adding burden to transfer object data to different form.
- Avoids the object data serialization/de-serialization for data retrieval, caching and storage.
- Reduces the consumption of on-heap memory and in turn to reduce and stabilize Java Garbage Collection (GC) pauses for latency sensitive applications.
- Overcomes current limitations of Java GC to manage much larger memory resources for massive dataset processing and computing.
- Supports the migration data usage model from traditional NVMe/SSD/HD to non-volatile memory with ease.
- Uses lazy loading mechanism to avoid unnecessary memory consumption if some data does not need to use for computing immediately.
- Bypasses JNI call for the interaction between Java runtime application and its native code.
- Provides an allocation aware auto-reclaim mechanism to prevent external memory resource leaking.

Background

Big Data and Cloud applications increasingly require both high throughput and low latency processing. Java-based applications targeting the Big Data and Cloud space should be tuned for better throughput, lower latency, and more predictable response time. Typically, there are some issues that impact BigData applications' performance and scalability:

1) The Complexity of Data Transformation/Organization: In most cases, during data processing, applications use their own complicated data caching mechanism for SerDes data objects, spilling to different storage and eviction large amount of data. Some data objects contains complex values and structure that will make it much more difficulty for data organization. To load and then parse/decode its datasets from storage consumes high system resource and computation power.

2) Lack of Caching, Burst Temporary Object Creation/Destruction Causes Frequent Long GC Pauses: Big Data computing/syntax generates large amount of temporary objects during processing, e.g. lambda, SerDes, copying and etc. This will trigger frequent long Java GC pause to scan references, to update references lists, and to copy live objects from one memory location to another blindly.

3) The Unpredictable GC Pause: For latency sensitive applications, such as database, search engine, web query, real-time/streaming computing, require latency/request-response under control. But current Java GC does not provide predictable GC activities with large on-heap memory management.

4) High JNI Invocation Cost: JNI calls are expensive, but high performance applications usually try to leverage native code to improve performance, however, JNI calls need to convert Java objects into something that C/C++ can understand. In addition, some comprehensive native code needs to communicate with Java based application that will cause frequently JNI call along with stack marshalling.

Mnemonic project provides a solution to address above issues and performance bottlenecks for structured data processing and computing. It also simplifies the massive data handling with much reduced GC activity.

Rationale

There are strong needs for a cohesive, easy-to-use non-volatile programing model for unified heterogeneous memory resources management and allocation. Mnemonic project provides a reusable and flexible framework to accommodate other special type of memory/block devices for better performance without changing client code.

Most of the BigData frameworks (e.g., Apache Spark™, Apache™ Hadoop®, Apache HBase™, Apache Flink™, Apache Kafka™, etc.) have their own complicated memory management modules for caching and checkpoint. Many approaches increase the complexity and are error-prone to maintain code.

We have observed heavy overheads during the operations of data parse, SerDes, pack/unpack, code/decode for data loading, storage, checkpoint, caching, marshal and transferring. Mnemonic provides a generic in-memory persistence object model to address those overheads for better performance. In addition, it manages its in-memory persistence objects and blocks in the way that GC does, which means their underlying memory resource is able to be reclaimed without explicitly releasing it.

Some existing Big Data applications suffer from poor Java GC behaviors when they process their massive unstructured datasets. Those behaviors either cause very long stop-the-world GC pauses or take significant system resources during computing which impact throughput and incur significant perceivable pauses for interactive analytics.

There are more and more computing intensive Big Data applications moving down to rely on JNI to offload their computing tasks to native code which dramatically increases the cost of JNI invocation and IPC. Mnemonic provides a mechanism to communicate with native code directly through in-place object data update to avoid complex object data type conversion and stack marshaling. In addition, this project can be extended to support various lockers for threads between Java code and native code.

Initial Goals

Our initial goal is to bring Mnemonic into the ASF and transit the engineering and governance processes to the "Apache Way." We would like to enrich a collaborative development model that closely aligns with current and future industry memory and storage technologies.

Another important goal is to encourage efforts to integrate non-volatile programming model into data centric processing/analytics frameworks/applications, (e.g., Apache Spark™, Apache HBase™, Apache Flink™, Apache™ Hadoop®, Apache Cassandra™, etc.).

We expect Mnemonic project to be continuously developing new functionalities in an open, community-driven way. We envision accelerating innovation under ASF governance in order to meet the requirements of a wide variety of use cases for in-memory non-volatile and volatile data caching programming.

Current Status

Mnemonic project is available at Intel's internal repository and managed by its designers and developers. It is also temporary hosted at Github for general view https://github.com/NonVolatileComputing/Mnemonic.git

We have integrated this project for Apache Spark™ 1.5.0 and get 2X performance improvement ratio for Spark™ MLlib k-means workload and observed expected benefits of removing SerDes, reducing total GC pause time by 40% from our experiments.

Meritocracy

Mnemonic was originally created by Gang (Gary) Wang and Yanping Wang in early 2015. The initial committers are the current Mnemonic R&D team members from US, China, and India Big Data Technologies Group at Intel. This group will form a base for much broader community to collaborate on this code base.

We intend to radically expand the initial developer and user community by running the project in accordance with the "Apache Way." Users and new contributors will be treated with respect and welcomed. By participating in the community and providing quality patches/support that move the project forward, they will earn merit. They also will be encouraged to provide non-code contributions (documentation, events, community management, etc.) and will gain merit for doing so. Those with a proven support and quality track record will be encouraged to become committers.

Community

If Mnemonic is accepted for incubation, the primary initial goal is to transit the core community towards embracing the Apache Way of project governance. We would solicit major existing contributors to become committers on the project from the start.

Core Developers

Mnemonic core developers are all skilled software developers and system performance engineers at Intel Corp with years of experiences in their fields. They have contributed many code to Apache projects. There are PMCs and experienced committers have been working with us from Apache Spark™, Apache HBase™, Apache Phoenix™, Apache™ Hadoop® for this project's open source efforts.

Alignment

The initial code base is targeted to data centric processing and analyzing in general. Mnemonic has been building the connection and integration for Apache projects and other projects.

We believe Mnemonic will be evolved to become a promising project for real-time processing, in-memory streaming analytics and more, along with current and future new server platforms with persistent memory as base storage devices.

Known Risks

Orphaned products

Intel's Big Data Technologies Group is actively working with community on integrating this project to Big Data frameworks and applications. We are continuously adding new concepts and codes to this project and support new usage cases and features for Apache Big Data ecosystem.

The project contributors are leading contributors of Hadoop-based technologies and have a long standing in the Hadoop community. As we are addressing major Big Data processing performance issues, there is minimal risk of this work becoming non-strategic and unsupported.

Our contributors are confident that a larger community will be formed within the project in a relatively short period of time.

Inexperience with Open Source

This project has long standing experienced mentors and interested contributors from Apache Spark™, Apache HBase™, Apache Phoenix™, Apache™ Hadoop® to help us moving through open source process. We are actively working with experienced Apache community PMCs and committers to improve our project and further testing.

Homogeneous Developers

All initial committers and interested contributors are employed at Intel. As an infrastructure memory project, there are wide range of Apache projects are interested in innovative memory project to fit large sized persistent memory and storage devices. Various Apache projects such as Apache Spark™, Apache HBase™, Apache Phoenix™, Apache Flink™, Apache Cassandra™ etc. can take good advantage of this project to overcome serialization/de-serialization, Java GC, and caching issues. We expect a wide range of interest will be generated after we open source this project to Apache.

Reliance on Salaried Developers

All developers are paid by their employers to contribute to this project. We welcome all others to contribute to this project after it is open sourced.

Relationships with Other Apache Product

Relationship with Apache™ Arrow:
Arrow's columnar data layout allows great use of CPU caches & SIMD. It places all data that relevant to a column operation in a compact format in memory.

Mnemonic directly puts the whole business object graphs on external heterogeneous storage media, e.g. off-heap, SSD. It is not necessary to normalize the structures of object graphs for caching, checkpoint or storing. It doesn't require developers to normalize their data object graphs. Mnemonic applications can avoid indexing & join datasets compared to traditional approaches.

Mnemonic can leverage Arrow to transparently re-layout qualified data objects or create special containers that is able to efficiently hold those data records in columnar form as one of major performance optimization constructs.

Mnemonic can be integrated into various Big Data and Cloud frameworks and applications. We are currently working on several Apache projects with Mnemonic:
For Apache Spark™ we are integrating Mnemonic to improve:
a) Local checkpoints b) Memory management for caching c) Persistent memory datasets input d) Non-Volatile RDD operations The best use case for Apache Spark™ computing is that the input data is stored in form of Mnemonic native storage to avoid caching its row data for iterative processing. Moreover, Spark applications can leverage Mnemonic to perform data transforming in persistent or non-persistent memory without SerDes.

For Apache™ Hadoop®, we are integrating HDFS Caching with Mnemonic instead of mmap. This will take advantage of persistent memory related features. We also plan to evaluate to integrate in Namenode Editlog, FSImage persistent data into Mnemonic persistent memory area.

For Apache HBase™, we are using Mnemonic for BucketCache and evaluating performance improvements.

We expect Mnemonic will be further developed and integrated into many Apache BigData projects and so on, to enhance memory management solutions for much improved performance and reliability.

An Excessive Fascination with the Apache Brand

While we expect Apache brand helps to attract more contributors, our interests in starting this project is based on the factors mentioned in the Rationale section.

We would like Mnemonic to become an Apache project to further foster a healthy community of contributors and consumers in BigData technology R&D areas. Since Mnemonic can directly benefit many Apache projects and solves major performance problems, we expect the Apache Software Foundation to increase interaction with the larger community as well.

Documentation

The documentation is currently available at Intel and will be posted under: https://mnemonic.incubator.apache.org/docs

Initial Source

Initial source code is temporary hosted Github for general viewing: https://github.com/NonVolatileComputing/Mnemonic.git It will be moved to Apache http://git.apache.org/ after podling.

The initial Source is written in Java code (88%) and mixed with JNI C code (11%) and shell script (1%) for underlying native allocation libraries.

Source and Intellectual Property Submission Plan

As soon as Mnemonic is approved to join the Incubator, the source code will be transitioned via the Software Grant Agreement onto ASF infrastructure and in turn made available under the Apache License, version 2.0.

External Dependencies

The required external dependencies are all Apache licenses or other compatible Licenses Note: The runtime dependent licenses of Mnemonic are all declared as Apache 2.0, the GNU licensed components are used for Mnemonic build and deployment. The Mnemonic JNI libraries are built using the GNU tools.

maven and its plugins (http://maven.apache.org/) [Apache 2.0] JDK8 or OpenJDK 8 (http://java.com/) [Oracle or Openjdk JDK License] Nvml (http://pmem.io) [optional] [Open Source] PMalloc (https://github.com/bigdata-memory/pmalloc) [optional] [Apache 2.0]

Build and test dependencies:
org.testng.testng v6.8.17 (http://testng.org) [Apache 2.0] org.flowcomputing.commons.commons-resgc v0.8.7 [Apache 2.0] org.flowcomputing.commons.commons-primitives v.0.6.0 [Apache 2.0] com.squareup.javapoet v1.3.1-SNAPSHOT [Apache 2.0] JDK8 or OpenJDK 8 (http://java.com/) [Oracle or Openjdk JDK License]

Cryptography

Project Mnemonic does not use cryptography itself, however, Hadoop projects use standard APIs and tools for SSH and SSL communication where necessary.

Required Resources

We request that following resources be created for the project to use

Mailing lists

private@mnemonic.incubator.apache.org (moderated subscriptions) commits@mnemonic.incubator.apache.org dev@mnemonic.incubator.apache.org

Git repository

https://github.com/apache/incubator-mnemonic

Documentation

https://mnemonic.incubator.apache.org/docs/

JIRA instance

https://issues.apache.org/jira/browse/mnemonic

Initial Committers

- Gang (Gary) Wang (gang1 dot wang at intel dot com)
- Yanping Wang (yanping dot wang at intel dot com)
- Uma Maheswara Rao G (umamahesh at apache dot org)
- Kai Zheng (drankye at apache dot org)
- Rakesh Radhakrishnan Potty (rakeshr at apache dot org)
- Sean Zhong (seanzhong at apache dot org)
- Henry Saputra (hsaputra at apache dot org)
- Hao Cheng (hao dot cheng at intel dot com)

Additional Interested Contributors

- Debo Dutta (dedutta at cisco dot com)
- Liang Chen (chenliang613 at Huawei dot com)

Affiliations

- Gang (Gary) Wang, Intel

- Yanping Wang, Intel
- Uma Maheswara Rao G, Intel
- Kai Zheng, Intel
- Rakesh Radhakrishnan Potty, Intel
- Sean Zhong, Intel
- Henry Saputra, Independent
- Hao Cheng, Intel

Sponsors

Champion

Patrick Hunt

Nominated Mentors

- Patrick Hunt <phunt at apache dot org> - Apache IPMC member

- Andrew Purtell <apurtell at apache dot org > - Apache IPMC member
- James Taylor <jamestaylor at apache dot org> - Apache IPMC member
- Henry Saputra <hsaputra at apache dot org> - Apache IPMC member

Sponsoring Entity

Apache Incubator PMC