2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing

An MPI-IO Compliant Java based Parallel I/O Library

Ammar Ahmad Awan[1], Muhammad Bilal Amin[1], Shujaat Hussain[1], Aamir Shafi[2], Sungyoung Lee[1]

[1]Department of Computer Engineering, Kyung Hee University, South Korea
{ammar, bilal, shujaat.hussain, sylee}@oslab.khu.ac.kr
[2]SEECS, National University of Sciences and Technology, Pakistan
aamir.shafi@seecs.edu.pk

***Abstract*— MPI provides high performance parallel file access API called MPI-IO. ROMIO library implements MPI-IO specifications thus providing this facility to C and Fortran programmers. Similarly, object-oriented languages such as Java and C# have adapted MPI specifications and their implementations provide HPC facility to its programmers. These implementations, however, lack parallel file access capability which is very important for large-scale parallel applications. In this paper, we propose a Java based parallel file access API called MPJ-IO and describe its reference implementation. We describe design details and performance evaluation of this implementation. We use JNI calls in our code to utilize functions from ROMIO library. In addition, we highlight the reasons for using JNI calls in our code.**

Keywords--Parallel I/O in Java; mpiJava; MPI-I/O; ROMIO

I. INTRODUCTION

Java is a well-known and widely used programming language because of its inherent features like multi-threading, platform independence, automatic garbage collection and dynamic type detection. Java has been adopted by the HPC community - an area which is largely dominated by C and Fortran - as an alternate language. Since Java's inception, there has been a growing interest to develop messaging middleware for Java language inspired by Message passing interface (MPI) libraries such as MPICH2 [1] and OpenMPI [2]. This led to the formation of Java Grande Forum and hence the first adaptation of MPI standard in Java called mpiJava 1.2 API [3]. Implementations of the mpiJava 1.2 API include, mpiJava (the software from HPJava group), F-MPJ, MPJ/Ibis and MPJ Express. We note that mpiJava (the software) is a native implementation of mpiJava1.2 API and is not to be confused with mpiJava 1.2 API itself.

The primary focus of prior research has been to prove that Java HPC implementations are suitable for developing real-world scientific applications. Our focus, however, is to enable parallel file access facility for these Java HPC libraries and provide a reasonable case by first developing an API which adapts the MPI-IO API [4] and then providing a reference implementation for this API. In this paper, first we describe how we have modeled the MPJ-IO API. Second, we describe two possible approaches for implementing this API. Third, we explain one of them in detail and provide its design and implementation details. In addition, we highlight the reasons for choosing the selected approach.

II. PROPOSED MPJ-IO API

The MPJ-IO API has been developed to guide the implementation experts take advantage of a standard API and implement the parallel file access facility for Java HPC libraries in a uniform fashion. The API is an adapted version of the MPI-IO API and considers the object-oriented nature of Java language. A full explanation of this API is beyond the scope of this paper but a draft version can be reviewed for better understanding[1].

III. REFERENCE IMPLEMENTATION

There are two approaches to implement the MPJ-IO API; first is to implement the API purely in Java language and second is to write Java Native Interface (JNI) wrappers to the ROMIO [5] library. ROMIO is a popular and most widely used MPI-IO implementation. We call the first approach as pure Java approach and second as wrapper approach.

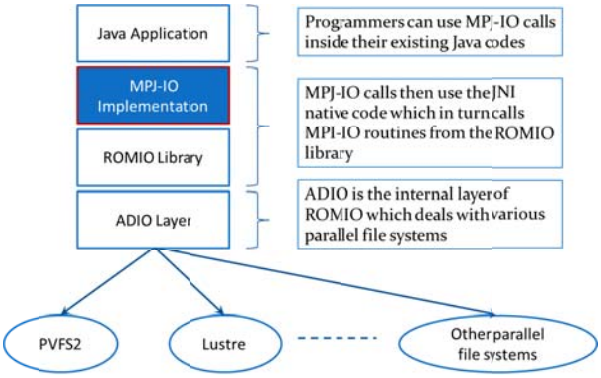

Figure 1. The Implementation Stack for MPJ-IO API

We first investigated the possibility of developing a pure Java implementation. We started with a performance evaluation of Java NIO based approaches for parallel file I/O in [6]. We faced two issues during the development of pure Java approach. First issue was the inconsistent performance of our pure Java implementation prototype against ROMIO. Second issue stems from the complexity of the parallel file system layer in the I/O stack - shown in Figure 1 - that poses a unique challenge for a pure Java implementation. The challenge is that most parallel file systems have a C/C++ interface which forces the developers

[1] Draft available from : http://hpc.seecs.edu.pk/~ammar/

978-0-7695-4996-5/13 $26.00 © 2013 IEEE
DOI 10.1109/CCGrid.2013.29

to first develop a JNI wrapper for each of them and then implement their parallel I/O API on top of it. This difficulty of implementation may render the pure Java approach almost useless but we are not concluding this at the moment. This will be investigated in the extension of this work.

We have developed the JNI wrapper approach using the widely accepted mpiJava software. We note that it is possible to implement the wrappers as a stand-alone library as well. The mpiJava software currently works on 32-bit platforms only but we plan to port it to 64-bit platforms in future. A notable benefit of JNI wrapper implementation is that it can be adopted by the HPC community easily compared to the pure Java approach. We note that HPC community easily embraces codes that are built on top of well-known implementations with proven performance [7].

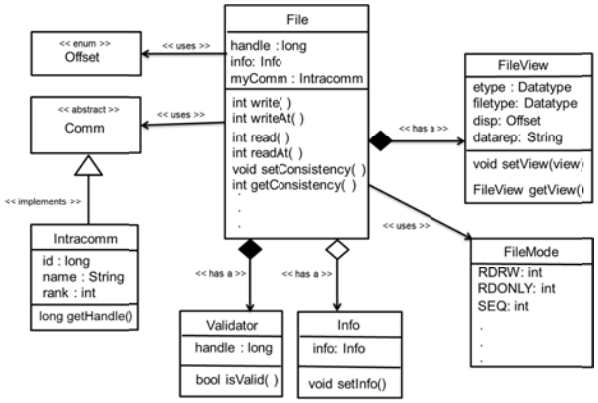

Figure 2. Class Diagram for the Wrapper Approach

The design of our wrapper implementation closely follows the mpiJava software. We modified some existing classes (`Intracomm and MPI`) in mpiJava and added some new MPJ-IO specific classes to it. The class diagram for wrapper approach in Figure 2 illustrates the interaction of the new `File, FileView, FileMode` and `Info` classes with the existing `Intracomm` and `Comm` classes. The methods' signatures for each of the classes have been explained in the draft of MPJ-IO API. Methods of `File` and other classes call native methods via JNI. The JNI code receives data from the Java layer, modifies and adjusts data-types and calls the respective C functions from ROMIO library. The methods for writing and reading files based on implicit offsets are shown below.

```
public Status read(Object buf, int bufOffset,
    int count, Datatype datatype) {
    // calls the native method here
  }

public Status write(Object buf, int bufOffset,
    int count, Datatype datatype) {
    // calls the native method here
  }
```

Our preliminary performance evaluation suggests a minimum 2x overhead for the JNI wrappers. We conducted our tests on a two-node cluster configured with PVFS2 parallel file system on Ethernet connection. We wrote a `Perf.java` file following the `perf.c` file from ROMIO. The resulting I/O bandwidth for the Java code shows 2X slower performance than its C counterpart. As our wrapper implementation is a work-in-progress, we are currently investigating performance optimizations to reduce this overhead. An extensive evaluation is also under progress.

IV. Related Work

Parallel Java (PJ) library has been recently updated with support for performing parallel I/O operations using the system called Parallel Datastore System (PDS) [8].

Software called jExpand (Java Expandable Parallel File System) [9] is another library for high-performance Java computing in heterogeneous distributed environments.

A large body of related works exists but majority of them either lack MPI-IO compliance or they are un-available for download.

V. Conclusions and Future Work

We proposed a parallel I/O API for the Java language inspired by MPI-IO API. We explored two approaches to implement this API and presented the second one called wrapper approach in detail. We presented the reasons for selecting the wrapper approach and highlighted its architecture and design. We also briefly mention about the performance of wrapper implementation on a two node cluster that uses PVFS2 parallel file system over Ethernet.

Acknowledgment

This research was supported by the MKE Korea, under the ITRC support program supervised by the NIPA (NIPA-2012-(H0301-12-2001)) and a grant from NIPA in 2013 (Global IT Talents Program).

References

[1] MPICH2 Project, http://www.mcs.anl.gov/research/projects/mpich2/

[2] OpenMPI Project, http://www.open-mpi.org/

[3] Carpenter, Bryan; Fox, Geoffrey; Ko, Sung-Hoon; and Lim, Sang, "mpiJava 1.2: API Specification" (1999). Northeast Parallel Architecture Center. Paper 66. http://surface.syr.edu/npac/66

[4] William Gropp, Ewing Lusk, and Rajeev Thakur. Using MPI-2: Advanced Features of the Message-Passing Interface. MIT Press, Cambridge, MA, 1999.

[5] ROMIO: A High-Performance, Portable MPI-IO Implementation, http://www.mcs.anl.gov/research/projects/romio/

[6] Ammar Ahmad Awan et al, "Towards Efficient Support for Parallel I/O in Java HPC", The Thirteenth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT2012), Bejing, China, December 14-16, 2012.

[7] Basili, V.R et al. , "Understanding the High-Performance-Computing Community: A Software Engineer's Perspective," Software, IEEE , vol.25, no.4, pp.29-36, July-Aug. 2008, doi: 10.1109/MS.2008.103

[8] Omonbek Salaev, Parallel Datastore System for Parallel Java, A Capstone Project Final Report, January 2010

[9] Jose M. Perez et al, "High performance Java input/output for heterogeneous distributed computing," Proc. 10th IEEE Symp. on Computers and Communications (ISCC 2005), June 2005, pp. 969-974