Sim-Hadoop : 신뢰성 있고 효율적인 N-body 시뮬레이션을 위한 Hadoop 분산 파일 시스템과 병렬 I/O

아마드, 이승룡, 정태충
경희대학교 컴퓨터 공학과,
e-mail : {ammar, sylee}@oslab.khu.ac.kr, tcchung@khu.ac.kr

Sim-Hadoop : Leveraging Hadoop Distributed File System and Parallel I/O for Reliable and Efficient N-body Simulations

Ammar Ahmad Awan, Sungyoung Lee and Tae Choong Chung
Dept. of Computer Engineering, Kyung Hee University, Korea

Abstract

Gadget-2 is a scientific simulation code has been used for many different types of simulations like, Colliding Galaxies, Cluster Formation and the popular Millennium Simulation. The code is parallelized with Message Passing Interface (MPI) and is written in C language. There is also a Java adaptation of the original code written using MPJ Express called Java Gadget. Java Gadget writes a lot of checkpoint data which may or may not use the HDF-5 file format. Since, HDF-5 is MPI-IO compliant, we can use our MPJ-IO library to perform parallel reading and writing of the checkpoint files and improve I/O performance. Additionally, to add reliability to the code execution, we propose the usage of Hadoop Distributed File System (HDFS) for writing the intermediate (checkpoint files) and final data (output files). The current code writes and reads the input, output and checkpoint files sequentially which can easily become bottleneck for large scale simulations. In this paper, we propose Sim-Hadoop, a framework to leverage HDFS and MPJ-IO for improving the I/O performance of Java Gadget code.

1. Introduction

Scientific simulations are mostly used by astrophysicists to analyze scientific phenomenon. Given the large scale of such simulations, scientists turn to high performance computing (HPC) solutions such as Message Passing Interface (MPI) parallelization of their sequential codes. HPC has made the life of physicists far simpler than it was before because of the sheer performance gains offered by them over conventional computers. The data analysis tasks that took months and years to complete now only take weeks, days or even hours because of massively parallel computers available to the scientific community.

Gadget-2 [1] is an open source production code developed by Volker Springel. He worked at the Max Planck Institute for Astrophysics. The main reference documentation for Gadget-2 is the paper written by Volker named "The Cosmological Simulation Code – Gadget2". It provides the implementation in C language and has been parallelized using MPI. It simulates very large system with help of massively parallel computers. It uses two methods namely Barnes Hut and TreePM. First is the pure Barnes Hut approach that has been discussed earlier whereas TreePM is a combination of Barnes Hut and Particle Mesh methods. In this project we will focus on Barnes Hut only.

Various computer scientists have argued that Java could make an excellent language for developing scientific codes. To date this argument has not convinced too many practicing computational scientists. The scarcity of high-profile number-crunching codes implemented in Java does not help the case. To help establish the practicality of real scientific computing using message passing Java authors have ported the parallel cosmological simulation code, Gadget-2, from C to Java, using MPJ Express. Java Gadget code writes a large amount of checkpoint/restart data to files after every few time-steps.

The current Gadget-2 Java [2] code does not use any specific parallel I/O method for handling large amounts of checkpoint data, instead, it uses the naïve approach of gathering all the data on Process 0 and writes the file sequentially to the disk. This can get some improvement if we use a parallel I/O approach and write it to a distributed or parallel file system. Parallel I/O has been standardized by MPI-IO [3] interface which has implementations in C, C++ and now in Java language as well. We envisage using MPJ-IO [4] library recently developed by our research group and use it inside Java Gadget code to write the checkpoint/restart files in a parallel fashion. Moreover, we believe that Java Gadget can use a fault tolerant distributed file system such as

Hadoop Distributed File System (HDFS). This will add reliability to the checkpoint data and allow the program to be restarted from the last failed position.

2. Sim-Hadoop

The proposed system will have two main deliverables; the first one will be the modified Java Gadget code which will use MPJ-IO library for writing checkpoint/restart files and second one will be the bridge code written to connect the Java code with the HDFS. HDFS is a distributed file system designed to run on commodity hardware. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS has master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on.
Sim-Hadoop will provide Java Gadget modifications. The idea is to focus on the sequential I/O part in the code and convert it so that it can use MPJ-IO methods. The underlying MPJ-IO implementation will be responsible to handle the parallelization of I/O requests. Figure 1 shows the development stack. It is clear that the Gadget-2 application will only utilize the important optimizations of the underlying I/O Middleware.

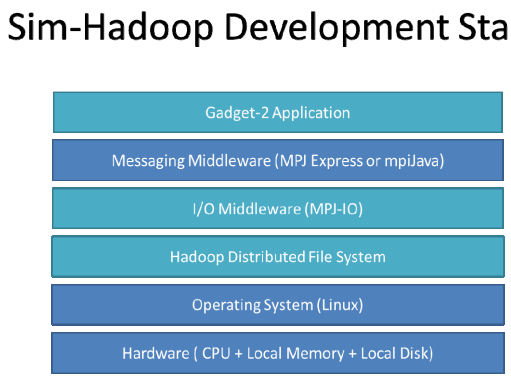

Figure 1 : Sim-Hadoop Development Stack

3. Hadoop-MPJ-IO Bridge

HDFS does not expose a POSIX interface so we will need to write a bridge code which allows the MPJ-IO library to connect to HDFS layer. This will be a novel contribution in addition to the modifications to the Java Gadget code for adding parallel I/O logic.

The *Hadoop-MPJ-IO Bridge* will be responsible to write the checkpoint file on distributed file system and make it available for the Java Gadget code to use it directly. We will need to set up this Bridge as a generic stand-alone code so that other researchers who want to use MPI-IO or MPJ-IO with HDFS can use it. This will be a great help to those programmers who do it manually in their codes.

4. Conclusion

In this paper, we proposed a novel strategy to improve performance and reliability of scientific simulations. The Java Gadget code provides a reasonably good example for our strategy since it writes a large amount of checkpoint data after every few time-steps. We propose and envisage developing a Hadoop-MPJ-IO bridge and use MPJ-IO calls inside the Java Gadget code. With minimal changes, the code's I/O performance improves dramatically and we believe that such methodology can be generally used for all Java based HPC applications. We plan to provide a complete implementation and performance study of this work in future. The code will be made publicly available.

Acknowledgement

This work (Grants No. 00048272) was supported by Business for Cooperative R&D between Industry, Academy, and Research Institute funded Korea Small and Medium Business Administration in 2011.

References

[1] Volker Springel, "The Cosmological Simulation Code – Gadget-2", Mon.Not.Roy.Astron.Soc. 364 (2005) 1105-1134

[2] Mark Baker, Bryan Carpenter and Aamir Shafi, "MPJ Express Meets Gadget: Towards a Java Code for Cosmological Simulations", Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science Volume 4192, 2006, pp 358-365

[3] Rajeev Thakur, William Gropp, and Ewing Lusk, "Optimizing Noncontiguous Accesses in MPI-IO," Parallel Computing, (28)1:83-105, January 2002

[4] Ammar Ahmad Awan, Muhammad Bilal Amin, Shujaat Hussain, Aamir Shafi and Sungyoung Lee, "An MPI-IO Compliant Java based Parallel I/O Library", IEEE/ACM CCGRID 2013, Netherlands, Delft.