Procedia Computer Science

Volume 51, 2015, Pages 2678–2682

ICCS 2015 International Conference On Computational Science

MPJ Express Meets YARN: Towards Java HPC on Hadoop Systems

Hamza Zafar[1], Farrukh Aftab Khan[1], Bryan Carpenter[2], Aamir Shafi[1] and Asad Waqar Malik[1]

[1] SEECS, National University of Sciences and Technology (NUST), Pakistan
{11bscshzafar, farrukh.khan, aamir.shafi, asad.malik}@seecs.nust.edu.pk
[2] School of Computing, University of Portsmouth, UK,
bryan.carpenter@port.ac.uk

Abstract

Many organizations—including academic, research, commercial institutions—have invested heavily in setting up High Performance Computing (HPC) facilities for running computational science applications. On the other hand, the Apache Hadoop software—after emerging in 2005—has become a popular, reliable, and scalable open-source framework for processing large-scale data (Big Data). Realizing the importance and significance of Big Data, an increasing number of organizations are investing in relatively cheaper Hadoop clusters for executing their mission critical data processing applications. An issue here is that system administrators at these sites might have to maintain two parallel facilities for running HPC and Hadoop computations. This, of course, is not ideal due to redundant maintenance work and poor economics. This paper attempts to bridge this gap by allowing HPC and Hadoop jobs to co-exist on a single hardware facility. We achieve this goal by exploiting YARN—Hadoop v2.0—that de-couples the computational and resource scheduling part of the Hadoop framework from HDFS. In this context, we have developed a YARN-based reference runtime system for the MPJ Express software that allows executing parallel MPI-like Java applications on Hadoop clusters. The main contribution of this paper is provide Big Data community access to MPI-like programming using MPJ Express. As an aside, this work allows parallel Java applications to perform computations on data stored in Hadoop Distributed File System (HDFS).

Keywords: Java MPI, MPJ Express, Apache Hadoop, YARN

1 Introduction

Since its emergence in 2005 the Apache Hadoop software [3] has become a popular, reliable, and scalable open-source distributed computing framework for processing very large datasets—so-called Big Data. A cornerstone of its success has been the easy-to-use MapReduce API [8]. Realizing the importance and significance of Big Data, many commercial and academic

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015

doi:10.1016/j.procs.2015.05.379

organizations have deployed compute clusters for running MapReduce jobs using the Hadoop framework. The "native" language for programming Hadoop is Java. This is far removed from the traditional High Performance Computing (HPC) paradigm, which employs massively parallel hardware including compute clusters equipped with low-latency and high-bandwidth interconnect. For programming such systems, the Message Passing Interface (MPI) standard [5] is the predominant API, usually available in C or Fortran. Driven by different goals or economics, a given organization may invest in either an HPC or a Hadoop cluster. If both paradigms are required—for example if some phase in the processing of a large Hadoop-resident data set is best expressed in terms of MPI programming—it may be necessary to maintain separate parallel facilities for running HPC and Hadoop computations. This, of course, is not ideal because this means higher capital and operational expenditure.

To avoid this extra cost, two possibilities suggest themselves. One is to allow Hadoop programs to be run on HPC clusters. Historically this was supported by the Hadoop On Demand (HOD) system, which allowed Hadoop MapReduce jobs to be scheduled by the traditional HPC resource manager Torque. Reference [7] contains a good account of the pros and cons of this approach. More recently the MR+ [1] system has allowed jobs using the Hadoop MapReduce API to be run under Open MPI. The second possibility—probably more attractive to organizations already heavily invested in the Hadoop model—is to allow running of "HPC" style applications on Hadoop clusters. Until relatively recently Hadoop was dedicated to MapReduce processing specifically, and this was hard to contemplate. But in 2012 Hadoop v2.0 was released and the landscape changed dramatically. The introduction of the YARN resource manager into the Hadoop stack has opened up the Hadoop environment a great deal, so that now MapReduce is just one framework that can execute under a YARN-managed cluster. Various other Apache Big Data projects (for example) have now targeted their platforms so that they can operate as computational frameworks under YARN, and thus execute harmoniously with MapReduce jobs in the same Hadoop cluster [4, 9, 2]. And of course this also opens the possibility of running "HPC" (or, more specifically, MPI-like) jobs under Hadoop.

The scope of this paper is limited to the second option, as we attempt to support execution of MPI-like Java programs on Hadoop clusters by providing a software solution encapsulated in a single library. The choice of Java looks unusual from a traditional HPC perspective, but this language is pervasive in the "Big Data" community. (We note that the importance of providing Java interfaces to MPI has also recently been recognized by the Open MPI developers [10].)

MPJ Express is an MPI-like—implements the mpiJava 1.2 API—messaging library with an active user community. In this paper, we discuss and present a new version of MPJ Express that integrates with YARN to allow executing parallel Java applications on a Hadoop cluster. This work mainly targets the Hadoop community that benefits from the introduction of new programming model/API to the Apache Hadoop stack. Our focus is towards producing and releasing an implementation that can be used by Hadoop community for conducting further research on this topic and improving scalability of Hadoop runtime system.

2 YARN Overview

This section provides an architectural overview of YARN—the resource scheduling layer in the Hadoop v2.0 software. As depicted in Figure 1, YARN consists of three main components: 1) a per cluster *central* ResourceManager (RM), a per node NodeManager (NM), and a per job ApplicationMaster (AM). A typical Hadoop cluster comprises of an instance of ResourceManager (RM)—running on a dedicated machine—along with a set of NodeManagers (NMs) executing on every compute node of the cluster.

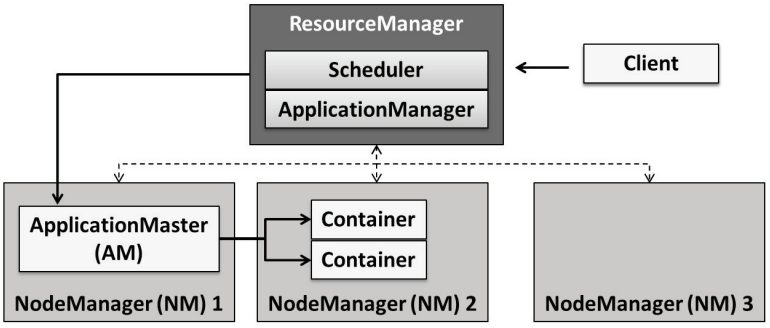


Figure 1: YARN Architecture

In a YARN-based Hadoop cluster, the RM module takes the central role in managing hardware resources and their allocation to user jobs. As the cluster boots, the NM module reports the capability—including number of processing cores and available memory—of its compute node to RM. Each NM continues to periodically communicate with RM to inform about the node status and current capability. This periodic communication is known as *heartbeat* between NM-RM and is shown as dotted lines in Figure 1. Note that all communication between NM-RM is achieved by appending data to these heartbeat messages.

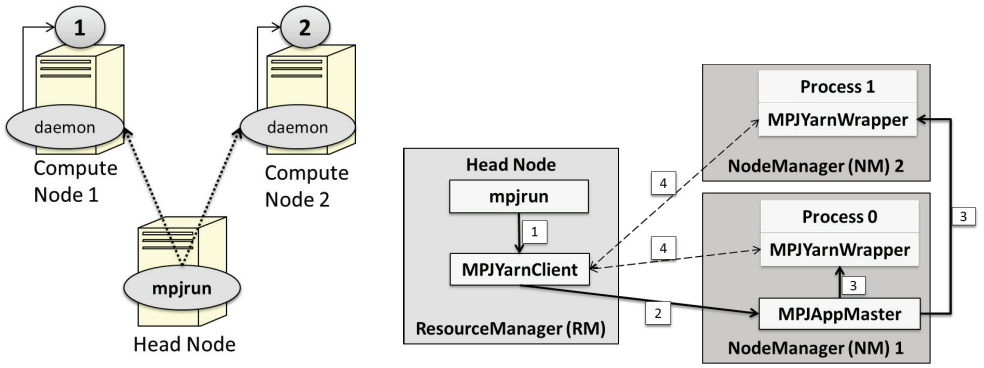
In order to execute a job, the end-user needs to develop a client application that is responsible for initiating a request to execute respective computational tasks on the cluster. Once the RM receives this request, it consults the built-in scheduler module and allocates a *container* on a cluster node within which the AM process for this particular client is executed. A container is the unit of resource allocation in the YARN framework, and is defined as "the logical bundle of resources including processing cores and memory". After this, the AM container negotiates allocation of required *worker* containers with the RM. In Figure 1, the AM process requests two additional containers that are allocated on Node 2 of the cluster.

3 Implementation/Evaluation of YARN-based Runtime

An important component of an MPI-like messaging system is the mechanism used to bootstrap processes across various platforms. The MPJ Express runtime provides a unified way of starting Java processes on a cluster or a network of computers. Figure 2(a) shows bootstrapping of an MPJ Express program on a cluster of one head node and two compute nodes. The runtime system consists of two modules: the `daemon` module that runs on compute nodes and the `mpjrun` module that runs on the head node.

We now shift our attention towards the YARN-based implementation of the MPJ Express runtime system. Unlike an HPC cluster, a Hadoop cluster is not equipped with shared file system. Instead it relies on a distributed filesystem called Hadoop Distributed File System (HDFS). For running code for Hadoop clusters, it is mandatory to upload application programs and related MPJ Express libraries on HDFS so that parallel processes running on compute nodes can access relevant code.

Figure 2(b) illustrates the implementation of the YARN-based runtime for the MPJ Express software on a Hadoop cluster with one head and two compute nodes. Note that the ResourceManager (RM) process executes on the head node, while NodeManager (NM) processes execute on compute nodes.



(a) Existing (b) YARN-based

Figure 2: The MPJ Express Runtime System

The parallel program execution is initiated at the head node of the cluster, where application developers invoke the `mpjrun` module. A typical command for this purpose is as follows:

```
mpjrun -np 4 -yarn hello.jar
```

Here, `-np 4` option specifies number of parallel process, `-yarn` specifies that the program is executed using the YARN-based runtime, and `hello.jar` is the user's application code.At this stage, the user code `hello.jar` exists on the local disk of the head node. Once executed, the `mpjrun` module in turn starts (arrow labeled 1) the `MPJYarnClient` module.

The `MPJYarnClient` module is responsible for submitting the user's application program to YARN. It does so by contacting the RM process, which responds by allocating a container for executing custom ApplicationMaster (AM)—`MPJAppMaster` in our case that runs on compute node 0 as shown in Figure 2(b) (arrow labeled 2). As reviewed in Section 2, any type of computation meant to run on a YARN-based Hadoop cluster requires an AM process. Note that YARN containers are typically described using a Container Launch Context (CLC). The `MPJYarnClient` module creates a CLC for the `MPJAppMaster` process, which includes an executable in the form of a Java ARchive (JAR) file and necessary commands to bootstrap the process within a container on a compute node. Note that the `MPJAppMaster` JAR file—named `mpj-app-master.jar`—is uploaded to HDFS by `MPJYarnClient`. The `MPJYarnClient` module also forwards total number of requested processes (`-np 4`) and the user's application code (`hello.jar`) CLI options to `MPJAppMaster`.

After booting up, `MPJAppMaster` communicates with the RM process to register itself. In response, the RM shares current maximum capabilities—in terms of nodes, processing cores, and available memory—of the YARN cluster. This information is used for making dynamic scheduling decisions, which is a unique feature of YARN. Note that `MPJAppMaster` is already aware of user's requirement for total number of parallel processes and now needs to map this to available resources. It also decides number of processing cores and amount of memory reserved for each container that hosts a single MPI process. Based on the number of required MPI processes, `MPJAppMaster` requests that many containers from the RM. The request includes number of cores and memory required for each container.

Once all the containers are allocated, `MPJAppMaster` creates a CLC for each container responsible for running user's program in parallel (arrows labeled 3). In this case CLC contains two JAR files—namely `mpj-yarn-wrapper.jar` and `hello.jar`—and commands to start the

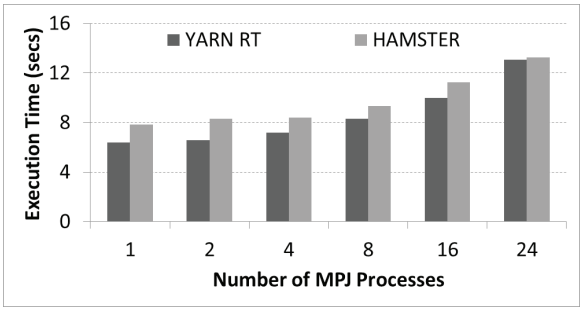


Figure 3: Performance Comparison of YARN-based Runtime (for MPJ) and HAMSTER

`MPJYarnWrapper` module. The `mpj-yarn-wrapper.jar` JAR file contains binary code for the `MPJYarnWrapper` module, while the `hello.jar` contains binary code for the user's application code. Note that these JAR files were previously uploaded to HDFS by `MPJYarnClient`. The `MPJYarnWrapper` process in turn launches the user code in a separate JVM using the Java's Reflection API. The main role of the `MPJYarnWrapper` is to send standard output and error streams of the user's application back to the `mpjrun` module (arrows labeled 4).

Figure 3 shows the bootstrapping time, in seconds, for MPJ Express YARN-based runtime and "Hadoop And MPI on the same cluSTER" (HAMSTER) project, which allows users to run MPI programs—executed using Open MPI—on Apache Hadoop platform. Currently HAMSTER is available as part of the Pivotal Hadoop Distribution (PHD) [6]. We define bootstrapping time as the "time for launching N parallel MPJ processes"—our implementation is marginally faster than HAMSTER.

References

[1] MR+: A Technical Overview. http://www.open-mpi.org/video/mrplus/Greenplum_RalphCastain-1up.pdf, 2012. [accessed 1-December-2014].

[2] Apache Giraph. http://giraph.apache.org/, 2014. [accessed 14-November-2014].

[3] Apache Hadoop. http://hadoop.apache.org, 2014. [accessed 14-November-2014].

[4] Apache Spark. https://spark.apache.org/, 2014. [accessed 14-November-2014].

[5] Message Passing Interface specifications. http://www.mpi-forum.org/docs/docs.html, 2014.

[6] Pivotal HD: Overview of Apache Stack and Pivotal HD Enterprise Distribution Components. http://pivotalhd.docs.pivotal.io/doc/2010/OverviewofApacheStackandPivotalComponents.html, 2014. [accessed 1-December-2014].

[7] Doug Eadline Joseph Niemiec Arun C. Murthy, Vinod Kumar Vavilapalli and Jeff Markham. *Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2*. Addison Wesley, March 2014.

[8] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[9] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41(3):59–72, March 2007.

[10] Jeffrey M. Squyres Oscar Vega-Gisbert, Jose E. Roman. Design and Implementation of Java bindings in Open MPI. Technical Report Pre-print submitted to Parallel Computing, July 2014.