High Performance Parallel I/O Schemes for Irregular Applications on Clusters of Workstations *

Jaechun No[1], Jesús Carretero **, and Alok Choudhary ***

jno@ece.nwu.edu
Dept. of Electrical Engineering and Computer Science,
Syracuse University, USA

Abstract. Due to the convergence of the fast microprocessors with low latency and high bandwidth communication networks, clusters of workstations are being used for high-performance computing. In this paper we present the design and implementation of a runtime system to support irregular applications on clusters of workstations, called "Collective I/O Clustering". The system provides a friendly programming model for performing I/O in irregular applications on clusters of workstations, and is completely integrated with the underlying communication and I/O system. All the performance results were obtained on the IBM-SP machine, located at Argonne National Labs.

1 Introduction

Due to the convergence of the fast microprocessors with low latency and high bandwidth communication networks, such as ATM, Myrinet, or the Gigabit Ethernet, clusters of workstations are being increasingly used for solving large-scale parallel scientific applications in cost-effective way. Most of those applications have tremendous I/O requirements [10, 7], including checkpointing of large-scale data sets, and writing of periodical snapshots for further visualization. Furthermore, a large subset of those applications are *irregular* applications, where accesses to data are performed through one or more levels of indirection [12]. Sparse matrix computations, particle codes, and many CFD applications where geometries and meshes are described via indirections, exhibit this feature.

In this paper we present the design and implementation of our runtime system for clusters of workstations, *collective I/O clustering*. The I/O architecture of clusters of workstations usually relies on a set of I/O servers, having local disks, and a set of diskless nodes. The design of our runtime system fits this feature, as

* This work was supported in part by Sandia National Labs award AV-6193 under the ASCI program, and in part by NSF Young Investigator Award CCR-9357840 and NSF CCR-9509143.

** Arquitectura y Tecnología de Sistemas Informáticos, Universidad Politécnica de Madrid, Spain.

*** Electrical and Computer Engineering, Northwestern University, USA

we distinguish between two kind of processors: I/O servers and compute nodes. All I/O details, such as data exchange, data distribution, and collective I/O, are transparent to the application programmer.

The main objectives for the collective I/O clustering are as follows:

- *Provide flexibility needed to the various I/O configurations for a cluster of workstations.* The collective I/O clustering is designed to support two kinds of I/O configurations: in the first I/O configuration, all processors are clients and I/O servers, and in the second I/O configuration, a subset of processors will only be I/O servers.
- *Provide user-controllable stripe unit.* Appropriate declustering of I/O requests over I/O servers should be addressed to produce high performance I/O bandwidth [5] and has been successfully implemented in the several file systems [8, 1, 4]. In the collective I/O clustering, we use a user-controllable stripe unit which is specified by GF(Group Factor) in the file-creation time.
- *Provide compression facility.* Compression has been traditionally used to reduce disk space requirement [14], but recently it has been applied to parallel applications managing large arrays with the aim of reducing the total execution time [11, 9]. The collective I/O clustering combines compression facility to achieve two major goals: reducing disk space requirement, and reducing total execution time.

The rest of the paper is organized as follows: Section 2 presents an brief overview of the collective I/O clustering on an irregular application. Section 3 presents the implementation details of the collective I/O clustering operation. Section 4 presents the performance results on the IBM/SP machine located at Argonne National Labs. Finally, some conclusions are presented in section 5.

2 Motivation

Figure 1(a) describes a typical irregular application, where it sweeps all the edges of an unstructured mesh. In the application, an input mesh file is read, and then the edges and nodes are distributed over processors. We used block distribution to spread them to the processors. *no_of_edges_partitioned_per_proc* represents the number of edges partitioned to a single processor. In the nested loops, *edge[j].V1* and *edge[j].V2* are two nodes connected by an edge *edge[j]*. The reference pattern is specified by *edge[j].V1* and *edge[j].V2*, called *indirection array*, and also these values are used to access to a global array. X is a data array which contains the physical values associated with each node. In this application, a node has an array consisting of 3 doubles, and other 2 floats.

Figure 1(b) shows an example of the edge and node partitions by using block distribution. In the processor 0, *4,5,8* are the remote indirection elements whose physical values mush be fetched from processor 1 and 2. All the remote values are fetched before the computation. After the computations are finished, the data (physical values associated with a node) are written to a global array whose