ISBN 978-83-60810-22-4

DCFMS: A Chunk-Based Distributed File System for Supporting Multimedia Communication

Cosmin Marian Poteras
University of Craiova
Software Engineering Department
Bvd. Decebal 107, Craiova, 200440, Romania
Email: cpoteras@software.ucv.ro

Constantin Petrisor
University of Craiova
Software Engineering Department
Bvd. Decebal 107, Craiova, 200440, Romania
Email: costy_petrisor@yahoo.com

Mihai Mocanu
University of Craiova
Software Engineering Department
Bvd. Decebal 107, Craiova, 200440, Romania
Email: mmocanu@software.ucv.ro

Cristian Marian Mihaescu
University of Craiova
Software Engineering Department
Bvd. Decebal 107, Craiova, 200440, Romania
Email: mihaescu@software.ucv.ro

***Abstract*—It is well known that the main drawback of distributed applications that require high performance is related to the data transfer speed between system nodes. The high speed networks are never enough. The application has to come out with special techniques and algorithms for optimizing data availability. This aspect is increasingly needed for some categories of distributed applications such as computational steering applications, which have to permanently allow users to interactively monitor and control the progress of their applications. Following our previous research which was focused on the development of a set of frameworks and platforms for distributed simulation and computational steering, we introduce in this paper a new model for distributed file systems, supporting data steering, that is able to provide optimization for data acquisition, data output and load balancing while reducing the development efforts, improving scalability and flexibility of the system. Data partitioning is being performed at a logical level allowing multimedia applications to define custom data chunks like frames of a video, phrases of text, regions of an image, etc.**

I. Introduction

REAL time visualization and computational steering are key elements when running a category of applications known as distributed (discrete event) simulation [1] [2]. Generally, simulation refers to the numerical evaluation of a model. It is well known that running simulations on distributed high performance environments might become embarrassingly slow if the analyze phase is performed as a post-processing phase. A simulation has to be exhaustively executed for all input data sets and data can only be analyzed as a post-simulation phase, even if in some cases the process may reveal useless results from the beginning. Therefore, we focused our recent research towards the need to design a high performance distributed simulation framework [3] whose main goal is to optimize scientific simulations. Our framework uses the concept of state-machines for representing general purpose parallel processing tasks and allows the researcher to visualize, analyze and steer the ongoing simulation avoiding irrelevant areas of the simulation process.

The main performance bottleneck that we dealt with while developing the state machine based distributed system (SMBDS) was the data handling itself (acquiring data very fast, dealing with multiple data sources, controlling the network availability, a.o.). Another important aspect that we've noticed was that in many situations it was more efficient to migrate the processing task (state machine) to the host that actually holds the input data than acquiring the data throughout the network. For that we needed a way to query all nodes and find out where the data resides before migrating.

There are many categories of distributed processing applications that demand high data availability. In a distributed environment a set of nodes holds the input data for the entire system. Each node of the system might also become data holder (data storage) as it might output data needed by other nodes in their assigned processing. It comes naturally that the data flow is a crucial factor for achieving the desired performance. It is a nice to have feature that a distributed file system commonly offers. If data flow would be entirely handled at the application level, the entire development process would be significantly slowed down, the application's maintenance would be less flexible, and there would be important doubts on the data transfers efficiency. Obviously this is not a desirable solution for handling data flow in a distributed environment. Instead one could separate the data flow handling into a standalone module whose main role is to acquire, store and provide the data required by the application's processes in the most efficient way.

In this paper we describe the Distributed Chunks Flow Management System (DCFMS) that enables and supports data steering of distributed simulation applications.The system acts like a file system while it adds two new innovative features: logical partitioning and data awareness. Logical partitioning allows the application to define the how the files shall be splitted into chunks. This is very important for avoiding unnecessary transfers of the entire file while only a part of it is needed, instead the transfers fit exactly the application's needs. The data awareness allow the application to query information related to data location. Most of the times in distributed environments it is desirable to migrate processes towards data than the other way around. This feature allow the application to decide whether to send data towards processes or processes towards data.

The rest of the paper is organized as follows. Section II discusses briefly some related work. Section III-A introduces the new model of DCFMS, as a distributed file system. Section III-B focuses on a description of the support for distributed data flow and load balancing issues in DCFMS and gives some implementation details for Chunker classes. Preliminary performance results are overviewed in Section IV. Section V concludes the paper and outlines the future trends of development.

II. Related Work

In this section we will mention two of the most popular and widely used distributed data transfer systems which have similar components with the ones introduced in this paper: Apache Hadoop - HDFS and BitTorrent Protocol.

BitTorrent Protocol [4] is a file-sharing protocol designed by Bram Cohen used in distributed environments for transferring large amounts of data. The idea behind BitTorrent is to establish peer-to-peer data transfer connections between a group of hosts, allowing them to download and upload data inside the group simultaneously. The torrents systems that implement BitTorrent protocol use a central tracker that is able to provide information about peers holding the data of interest. Once this data reaches the client application, it tries to connect to all peers and retrieve the data of interest. However, it is up to the client to establish the upload and download priorities. Torrents systems might be a good choice for distributed environments, especially for those based on slower networks. However, the main disadvantages of torrent systems are related to the centralized nature of the torrents tracker as well as leaving the entire transfer algorithms and priorities up to the client application which might cause important delays if the transfers trading algorithm chooses to serve a peer that might have a lower priority at the application level. The reliability of the entire system is concentrated around the tracker; if the tracker goes down, the entire system becomes not functional. Torrents are mainly systems that transfer files in distributed environments in raw format without any logical partitioning of the data. Such logical partitioning might often prove to be very important. For example if an imaging application needs a certain rectangle of an image it would have to download the entire file and then extract the rectangle by itself instead of just downloading the rectangular area and avoid transferring unnecessary parts of the file. As a remarkable advantage of torrents systems we could mention self-sustainability [5] due to peer independence and redundancy.

Hadoop Distributed Files System (HDFS) has been designed as part of Apache Hadoop [6] distributed systems framework. Hadoop has been built on top of the Google's Map-Reduce architecture and HDFS. HDFS proved to be scalable, and portable. It uses a TCP/IP layer for internal communication and RPC for client requests. The HDFS has been designed to handle very large files that are sent across hosts in chunks. Data nodes can cooperate with each other in order to provide data balancing and replication. The file system depends closely on a central node, the name node whose main task is to manage information related to directory namespace. HDFS offers a very important feature for computational load balancing, namely it can provide data location information allowing the application to migrate the processing tasks towards data, than transferring data towards processing task over the network [7]. The main disadvantage of HDFS seems to be the centralized architecture built around the name node. Failure of the name node implies failure of the entire system. Though, there are available replication and recovering techniques of the name node, this might cause unacceptable delays in a high performance application.

III. Methods and Algorithms

A. *Conceptual Model for Distributed File System*

In this section the conceptual model of our distributed file system will be introduced. The system is simple, based on a client-server architecture. The entire model has been built around the key element, data chunk. The data chunk usually represents a file partition but none the less it can be any data object required by the application's processes. Besides the data piece itself, a data chunk also contains meta-information describing the data piece, like: size, location inside source file, the data type, timestamp of latest update or the class that handles chunks of its type.

Figure 1 illustrates the systems model. The most important contribution of DCFMS is that it handles chunks of different types in an abstract mode without actually knowing what is inside the chunk, leaving the data partitioning up to the application level. This is very important from the application's perspective as it can define the way files are partitioned into chunks and how they can be put together again to reconstruct the initial file allowing the application to map data chunks to processing tasks in the most appropriate way for efficient processing. No restrictions are imposed by the DCFMS on data partitioning.

The Type Manager is the bridge between the abstract representation of data chunks and their actual type. The Type Manager is able to make use of external classes where all the file type specific functionality can reside. The classes are dynamically loaded whenever the application layer needs partitioning, files reconstruction as well as information related to the collection of chunks (i.e. the number of chunks). It is the applications' developer task to implement the data chunks handler classes. The DCFMS only provides a set of interfaces that helps the developer implement the partitioning logic.

For example, one might need to handle two types of files in their distributed application: image files and text files. In case of the image files a data chunk might be represented by a rectangular region of the initial image. Multiple such

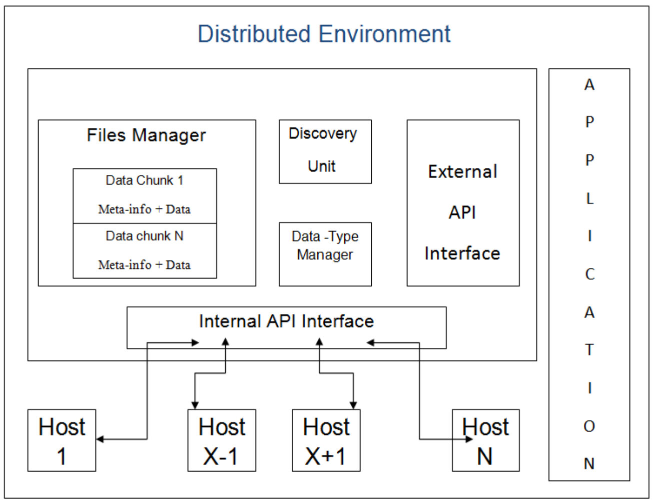

Fig. 1. DCFMS design model

chunks can cover the entire image. An image can be split into rectangular chunks by dynamically invoking the image partitioning method. In case a node needs an entire file that is spread all across the system, DCFMS can acquire all its chunks from different hosts and recompose the image by dynamically invoking the image reconstruction method. In case of a text file, the chunks can take the form of paragraphs, or pages, or simply an array of characters of a certain size. In a similar way the files can be dynamically partitioned and reconstructed.

Later in this paper we will discuss the development effort involved in writing such classes.

The proposed DCFMS is able to scale up dynamically at run time without using a central node. This functionality is achieved by the Discovery Unit which broadcasts and listens to discovery messages.

There are two API interfaces that allow DCFMS nodes to communicate with each other and also with the client application.

B. Distributed Concepts Support and Implementation

1) Data flow: For a better understanding of the data flow algorithm, we will analyze a concrete scenario. Lets assume DCFMS consists of nodes N0, N1, ..., Nn, and let node N0 be interested in aquiring data chunks C1, C2, ..., Cm. N0 will broadcast a request for C1, ..., Cm to the entire DCFMS. Nodes N1, ..., Nn reply back to N0 with a subset of C1, ..., Cm that they host. As soon as replys arrive, N0 builds a chunks availability matrix having as rows the nodes N1, ..., Nn and as columns chunks C1, ..., Cm. (Ni,Cj) gets valued 1 if the chunk Cj is available on host Ni, otherwise it gets valued 0. N0's main goal is to establish as many connections as possible, but not more than one connection per serving host (at most n-1 connections at a time). Chunks availability responses are performed in an asynchronous manner so that N0 won't have to wait for all responses before proceeding with transfers. Instead it will establish connections as the responses arrive, overlapping chunks transfer with availability requests. Whenever a chunk transfer completes, the External API will be informed about it and the client application can start processing the newly acquired data. As chunks might spread across DCFMS while N0 transfers it's chunks, the availability matrix will be constantly updated by sending new availability requests whenever a chunk transfer completes and N0 has established less than n-1 connections (free download slots available).

2) Support for load balancing: In distributed applications it often happens that the processing of a data chunk requires less time than the transfer of the data itself. For this reason it might be a good practice to migrate the processing task towards the data than transferring data to the processing host. The DCFMS is able to provide through its external API locating information about the data it holds (data aware system). It is the application's task to migrate the processing tasks throughout the nodes in order to reduce or eliminate the data transfer time.

3) Application developer's task: Implementing Chunker classes: Chunker classes define how files or data objects are split into data chunks. A chunker class is nothing else than a class that implements a Chunker interface defining the following methods:

- GetChunk(chunkId)
- IsChunkAvailable(chunkId)
- ReconstructFile(filename)

Chunker classes are dynamically invoked at run time every time chunks or their associated meta-data are being requested. Data chunks are mapped to chunker classes by their meta-data.

IV. Experimental results

We conducted a set of experiments for the preliminary performance evaluation of our DCFMS, as a standalone system, out of the scope of the framework in which it will be finally integrated. To evaluate DCFMS we've made use of two environments:

- A high performance Myrinet network of 4 Gbps bandwidth consisting of 8 identical hosts with Intel Core 2 Duo E5200 processors, 1GB of memory and the hard drive benchmarked at an average read speed of 57MB/s and write speed of 45MB/s.
- A regular Ethernet network of 100Mbps bandwidth consisting of 8 identical hosts 4 hosts with Intel Core 2 Quad processors and 4 GB of memory.

The purpose of the experiments was to determine how fast will perform the DCFMS in one-to-many and many-to-one scenarios. We've picked those two scenarios since they represent the worst and the best traffic demanding scenarios. In one-to-many scenario a certain file was hosted by one node and had to be transferred to all the other nodes starting simultaneous. The reverse work had to be performed in a many-to-one scenario, namely all hosts except one hosted the file, and they had to serve collectively the file to the client host. Each scenario has been run for different chunk

TABLE I
MYRINET ONE-TO-MANY RESULTS

Chunk Size	Min. time 7 hosts (s)	Max. time 7 hosts (s)	Avg. Time 7 hosts (s)
256KB	14.197	16.973	15.730
512KB	9.072	10.291	9.704
1MB	9.110	10.534	9.929
2MB	8.800	9.802	9.404
5MB	9.779	11.926	10.960
10MB	9.221	12.253	10.99
15 MB	9.166	11.529	10.353
20MB	12.444	18.583	15.640
25MB	13.763	18.407	17.066

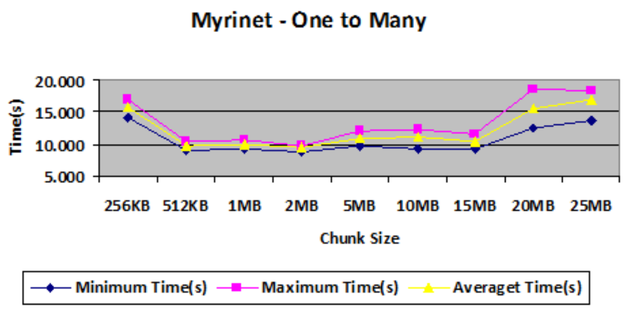

Fig. 2. Myrinet One-to-Many results

sizes. The test cases presented in this paper focus on the transfer speed of DCFMS rather than on the computational performance of a system based on it. Around 50 runs were performed for each case and the results were statistically processed, avoiding singularities. We appreciate that the results can be significantly improved by our DCFMS running in its real design environment instead of a testbed, by using the data awareness capabilities discussed in the previous sections.

A. *Myrinet Network results:*

Test case 1: One-to-Many one sender host and 7 receivers that request the same file of 180.6MB simultaneously. Results are presented in Table I and Figure 2

Test case 2: Many-to-One 7 senders that will serve one host that requests the same file of 180.6MB from all senders. Results are presented in Table II and Figure 3

By examining the results obtained at test cases 1 and 2 we can conclude that the chunk size has an important impact on the file system's performance. To obtain best timings, the Myrinet-based application developer has to choose a chunk size between 2MB and 5MB.

B. *Ethernet Network results:*

As the Ethernet speed is far less than the Myrinet network, we decided to use a small file, namely a 9.8MB file.

Test case 3: One-to-Many one sender host and 7 receivers that request the same file of 9.8MB simultaneously. Results are presented in Table III and Figure 4

TABLE II
MYRINET MANY-TO-ONE RESULTS

Chunk Size	Time (s)
256KB	11.362
512KB	5.693
1MB	5.466
2MB	4.168
5MB	3.692
10MB	4.111
15 MB	4.216
20MB	5.856
25MB	8.378

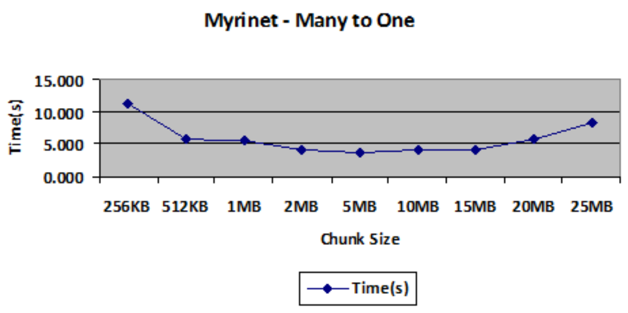

Fig. 3. Myrinet Many-to-One results

TABLE III
ETHERNET ONE-TO-MANY RESULTS

Chunk Size	128KB	256KB	512KB	1MB	2MB
Minimum time of the 7 hosts (s)	77.241	80.833	64.011	70.572	57.159
Maximum time of the 7 hosts (s)	83.553	85.908	80.121	82.930	75.547
Average Time of the 7 hosts (s)	80.344	83.550	72.994	77.346	67.021

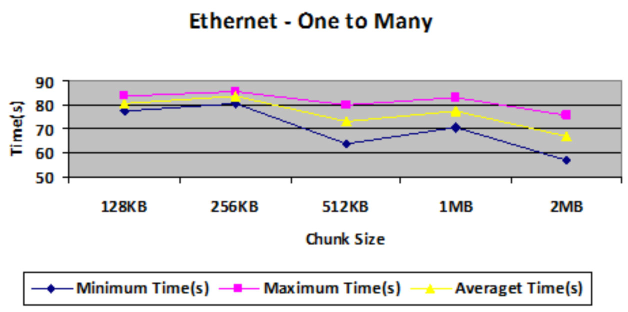

Fig. 4. Ethernet One-to-Many results

TABLE IV
ETHERNET MANY-TO-ONE RESULTS

Chunk Size	128KB	256KB	512KB	1MB	2MB
TIme (s)	29.697	29.048	28.940	28.726	28.752

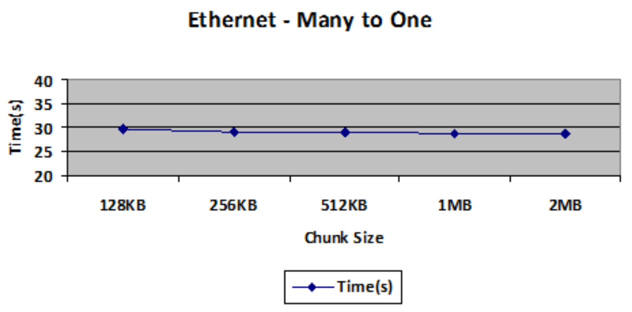

Fig. 5. Ethernet Many-to-One results

Test case 4: Many-to-One 7 senders that will serve one host that requests the same file of 9.8MB from all senders. Results are presented in Table IV and Figure 5

Unlike the Myrinet network, in case of the Ethernet the chunk size doesn't have a big impact on the performance. However the Ethernet network proved not to be the appropriate environment for high performance distributed applications that require high data availability.

V. CONCLUSIONS AND FUTURE WORK

In this paper we've introduced a new model of distributed files systems. The dynamic discovery feature of the system ensures the scalability of the system, the decentralized architecture improves the reliability, while the dynamic data handling offered by the chunker classes make the system flexible and easier to extend. Some other important features of the system, that worths mentioning are: load balancing support due to the data awareness feature and the ability to define chunks that can have any logical meaning.

Important contributions of the system relate to: custom logical partitioning defined at the application level (abstractly handling) and load balancing support due to the data awareness (data location information) feature while maintaining a high data availability.

The system shows good performance in very high speed networks (Myrinet), but it can also be a good choice in Ethernet networks for applications not requiring transfers of high data volumes across the network.

As future development of the system we could mention the hosts' speed ranking which could be very significant when deciding the source hosts, the network traffic monitoring which could help deciding the route that should be followed for a faster download and none the less the system needs caching techniques.

Being designed as part of a distributed simulation framework [3], as mentioned in the Introduction, DCFMS shall be able to provide support for computational steering. Besides steering the simulation processes the researcher shall be able to also steer data storage, or alter data held by DCFMS while simulation is running. Some algorithms for optimal probabilistic replication of data when the system is in idle state would also be a nice to have feature in our future developments.

REFERENCES

[1] R. J. Allan and M. Ashworth. A survey of distributed computing, computational grid, meta-computing and network information tools. Daresbury, Warrington WA4 4AD, UK,2001, pp. 38-42
[2] Esnard, A. Richart, N. , Coulaud, O. A Steering Environment for Online Parallel Visualization of Legacy Parallel Simulations,. Proceedings of DS-RT'06 - 10th IEEE International Symposium on Distributed Simulation and Real-Time Applications, 2006, pp.7-14
[3] Cosmin Poteras, Mihai Mocanu Grid-Enabled Distributed Simulation— A State Machine Based Approach, Proceedings of TELFOR 2010, Belgrade, Serbia, pp. 1323-1326
[4] Bram Cohen- The BitTorrent Protocol Specification, http://www.bittorrent.org/beps/bep_0003.html
[5] D. Menasche, A. Rocha, E. de Souza e Silva, R. M. Leao, D. Towsley, A. Venkataramani - Estimating Self-Sustainability in Peer-to-Peer Swarming Systems, Journal of Performance Evaluation Volume 67 Issue 11, November, 2010.
[6] http://hadoop.apache.org/
[7] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, James Majors, Adam Manzanares, and Xiao Qin - Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters, IPDPSW 2010, Atlanta, pp. 1-9