NSM: A Distributed Storage Architecture for Data-Intensive Applications

Zeyad Ali Qutaiba Malluhi

Distributed Computing Laboratory
Computer Science Department
Jackson State University
Jackson, MS 39217

zeyad.f.ali@jsums.edu qmalluhi@jsums.edu

Abstract:

Several solutions have been developed to provide data-intensive applications with the highest possible data rates. Such solutions tried to utilize the available network resources through parallel I/O and TCP/IP tuning in order to achieve a better data throughput. The focus was on achieving the highest possible data rate while other performance enhancements factors were ignored. Furthermore, most of those solutions were point solutions and designed to work in a specific environment for a particular application.

In this paper, we introduce the Network Storage Manager (NSM). NSM is a java-based, high-performance, distributed storage system with auto reconfigurability that has been developed in the Distributed Computing Laboratory at Jackson State University. The system is designed as a framework for data-intensive distributed applications of different natures. In addition to an architecture that employs parallelism, scalability, crash recovery, and portability, NSM provides applications with full control to optimize other application-controllable features by allowing applications to fine-tune such features or even plug-in their modules and use it instead of the standard NSM implementation.

1. Introduction

High-performance multimedia, visualization, and other data-intensive applications usually use datasets that are of huge sizes typically hundreds of Gbytes or even Terabytes. These datasets not only do not fit in the memory available for the client machine, but also may not fit on the local disk. Moreover, these data sets are most of the time available on remote machines, which imposes long delays and slow application startup.

However, large data sets are not usually needed as a whole by applications. A terrain visualization application is only interested, at any time, in a certain region of the huge image. A high-resolution distributed video player may be interested in the last minutes of a huge video clip, and it does not want to waste time and resources in downloading unwanted data.

Some traditional solution applications [1] [3] [4] segmented and distributed the huge datasets. In this case, the client only fetches the segments that are required to visualize the current region. These solutions are by far domain-specific solutions and cannot be utilized to provide similar solutions for other kinds of data-intensive applications.

Another attempt [5], in an effort to provide a general solution, suggested modifying operating systems to allow application control on virtual memory and paging. Unfortunately, it did not happen in practice because it could not justify changing the operating system in favor of a limited set of application.

Partitioning huge data object into small chunks and distributing them onto storage servers, adds some kind of parallelism that helps the client machine to achieve a better performance in handling the out-of-core data. But parallelism by itself does not guarantee the optimal performance of an application since higher data throughput does not necessarily result in better application performance. Other performance enhancement techniques such as proper data layout, prefetching, and cache replacement policies should be considered very supportive to better fulfill the needs of the application and optimize its performance. But because of the different natures of applications, an algorithm that is optimal for one application may not be so to other ones. Furthermore, features, not necessarily related to performance, such as data partitioning, transport protocols, meta data, security schemes and encoding algorithms are also believed to be application-dependent.

2. NSM Solution

In NSM, we provide a general solution that has an architecture in which all the high-performance features and services, needed to achieve high and reliable data throughput, are core system features. The system has a unique architecture that provides many advantages including high performance, reliability, self-healing, load

balancing and seamless access. Our system utilizes multiple parallel data streams to achieve load balancing and high data rates. The system offers users transparent and seamless access to the physically distributed datasets. Applications utilizing NSM for their data storage are smart applications because they automatically inherit all of its merits and features including, high availability, reliability and high throughput.

In the other hand, any feature that is believed to be application-dependant is designed as a pluggable system component with standard implementation. Such features may be replaced with an application-specific implementation by the application's developer. For example, if the standard round-robin implementation for the data layout of the object to be distributed, is not believed by the application's developer to be the optimal layout for that specific domain, he/she can simply implement his/her algorithm by implementing the NSM layout interface and plug-in the new implementation.

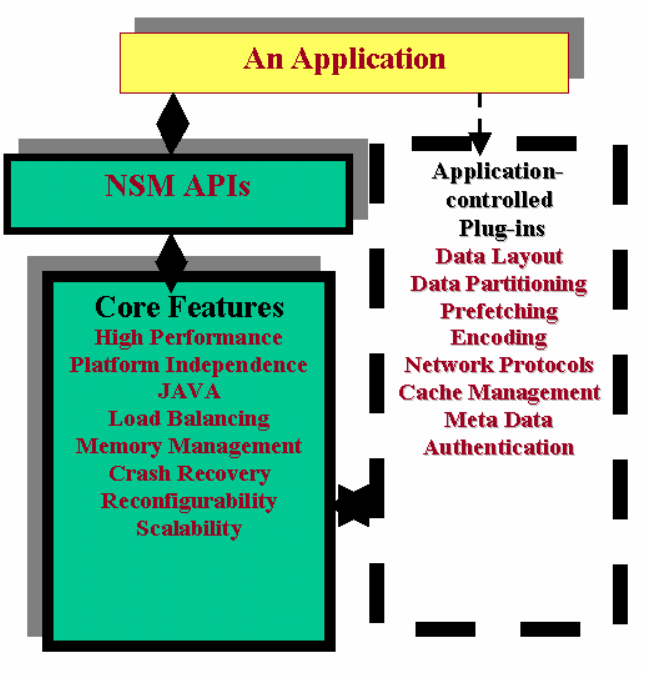

Figure 1. NSM architecture from an application's view

All the application-controllable features of NSM are designed in way that avoids interdependence between any two interacting or interrelated components; this design characteristic provides higher degree of flexibility to applications in the sense that plugging an application-specific algorithm, such as data partitioning, does not result in changing other controllable system features. However, application plug-ins should conform to the set of interfaces provided by the system.

In order to simplify using NSM by applications, we provide a group of APIs that can be used by applications. An application can use straightforward APIs such as ***writeDataSet*** **(), *openDataSet*** **(), *closeDataSet*** **().** If an application needs to plug its own implementation for any of the pluggable features of system, it can also specify the required plug-in by using commands such as ***setPrefetcher*** *(Prefetcher ClasName)*. Figure 1 illustrates NSM architecture from an application's point of view.

3. NSM System Overview

3.1. Data Layout over Storage Servers

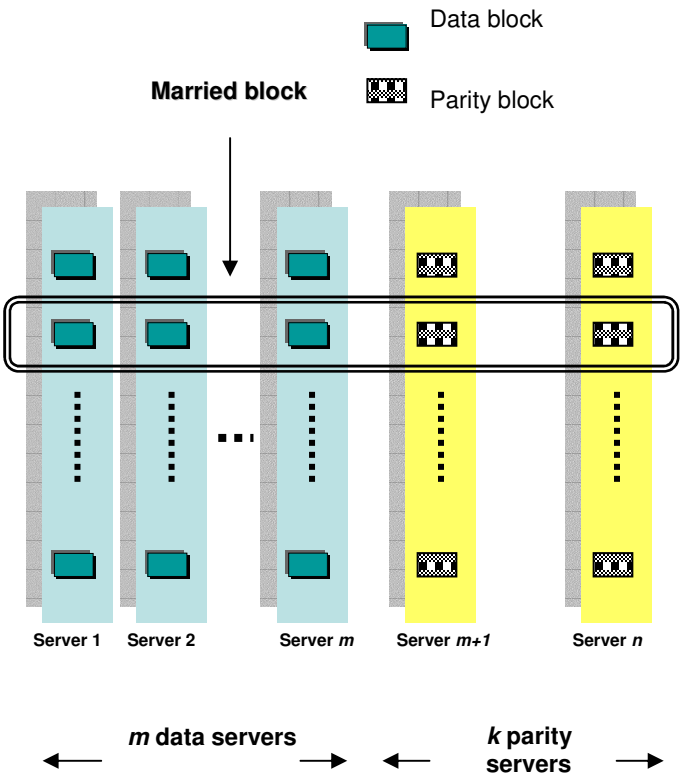

Figure 2. Data set layout over storage servers

In NSM we partition the data set into a number of small data blocks. The partitioning algorithm may be a standard fixed-size algorithm or an application-provided algorithm. The system distributes the blocks across multiple data servers. To enable dependable service, NSM uses coding to add redundancy to the original data. This redundancy enables applications to retrieve the original data even if portion of the data is unavailable due to server and/or network failure. The encoding/decoding algorithms are also NSM pluggable feature. The data blocks and their corresponding parity blocks are grouped in *married* blocks. A married block contains one block for each data and parity

server. Selecting the blocks in each married block is an application issue and depends on its decision on the suitable data layout. To ensure load balancing, NSM distributes the blocks of a married block to distinct servers. At the reader's side, a request to a single block is considered to be a request to all the blocks in that married block. See figure 2.

3.2. Distributing Data Sets

The married blocks are buffered for uploading. Buffering is essential to allow client machines to smoothly handle huge or remote data sources. Efficient data service can then be achieved by using multiple concurrent streams established between the client and the distributed data servers. After uploading all the blocks of the data source, the meta data is obtained from the layout algorithm and uploaded to one or more designated meta servers. Figure 3 illustrates the details of how applications can NSM features in distributing huge datasets.

3.3. Meta Data

Applications have full control over the meta data generated after distributing the dataset. They can add their own key-value pairs to the basic meta data generated by NSM. However, application can still use the detailed mode of meta data which results in a larger meta file. The fully detailed meta mode is needed where the logical or physical block names and/or sizes can not be algorithmatically reconstructed from the basic meta mode.

3.4. Data Retrieval

The system offers applications transparent and seamless access to the physically distributed data sets. Applications can use NSM as a high-performance random input stream. An application can open multiple data sets at a time using the same NSMReader. This feature is very advantageous for applications that need to work with multiple data sets of similar or even different types. A video player application [9] successfully played a video file by joining blocks from two data sets. A huge MPEG-1 video clip was distributed into two different data sets. One set contained all I and P frames and the second one had all B frames.

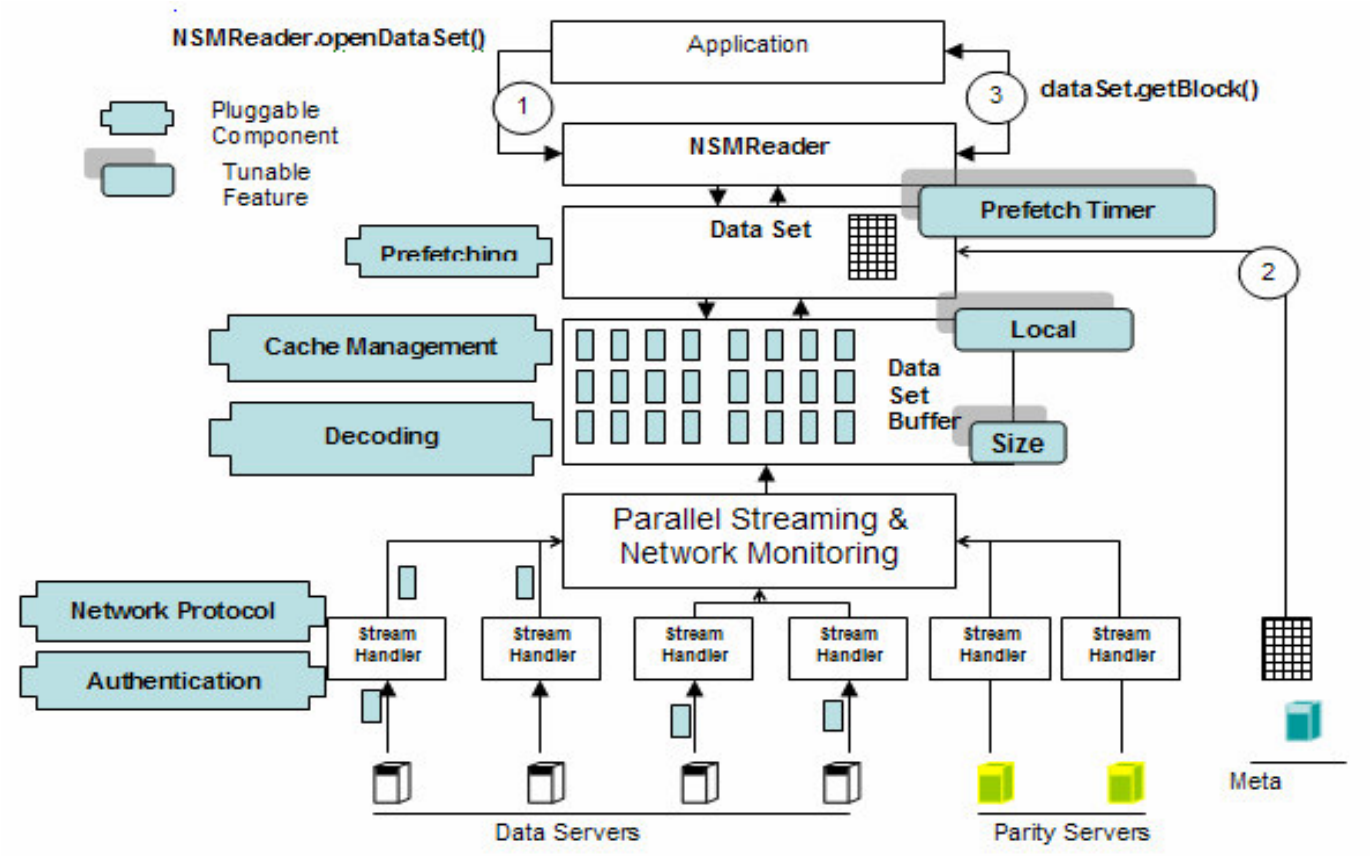

Figure 3. Distributing data sets using NSMWriter

Figure 4 illustrates NSM capability to provide applications with blocks from more than one dataset.

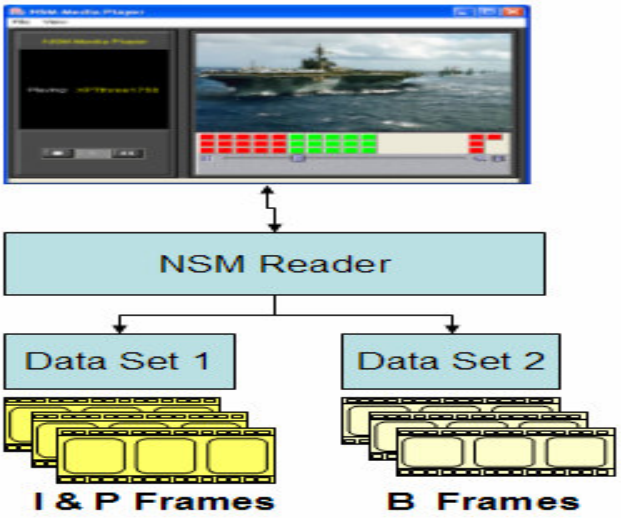

Figure 4. Handling multiple data sets in one input stream

Each data set has its own buffer. The buffer size is set by the application. A prefetching mechanism is utilized in an effort to have the most probably to be requested blocks in memory even before they are requested by the application.

The standard prefetching algorithm is used unless otherwise. The prefetcher does not start requesting blocks unless the application is not sending any request for a predefined period of time. Applications requests have higher priority over prefetching request. Applications can send requests with different levels of priority. A request to a single block results in requesting all the blocks in the corresponding married block. The requests are queued and served according to their priority. The application can void any queued requests; this feature may be helpful in case of sudden changes in locality.

If the data set buffer if full and more requests are coming, a cache management algorithm is used to decide which blocks to dispose. The standard cache replacement policy will dispose the least recently used block. However, since cache management is a pluggable feature, an application can plug-in its own implementation instead.

The blocks, as shown in figure 5, are downloaded in parallel from their storage servers using asynchronous system calls. The system recovers any server failure or network delay by transparently switching to any of the available parity servers. The missing data blocks are reconstructed by the decoding the corresponding parity blocks. The on-the-fly data recovery leads to a high reliability without sacrificing the performance.

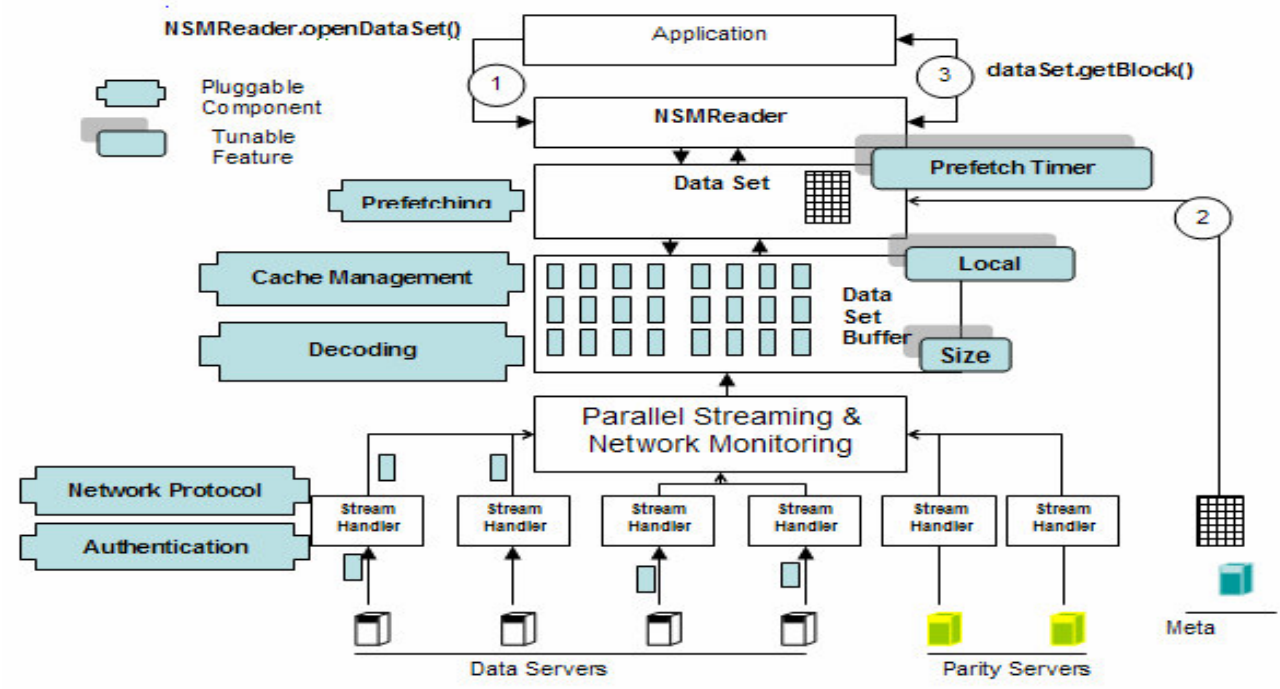

Figure 5. Handling applications requests by NSMReader.

4. Related Work

Related distributed mass storage technologies are DPSS [8], HPSS [7] and OceanStore [6]. DPSS is very vulnerable because it does not provide any mechanism for fault tolerance or recovery. HPSS is a hierarchical storage system that deals with different levels of storage devices such as disks and tapes. In HPSS, server mirroring (rather than coding) or automatic recovery from the tape is employed to achieve high availability. OceanStore addresses the storage utility model. It deals with a global storage environment consisting of thousands of un-trusted servers.

OceanStore data consists of nomadic encrypted objects with many floating replica. NSM, on the other hand, deals with a controllable pool of trusted servers. NSM blocks have specific locations because performance is a high priority.

5. Experimental Results

Several performance evaluation experiments were conducted to measure the impact of each of the high-performance features provided by NSM. One of them is the effect of the parallel data transfer on the achieved data rate while downloading a data set. The results are shown in figure 6.

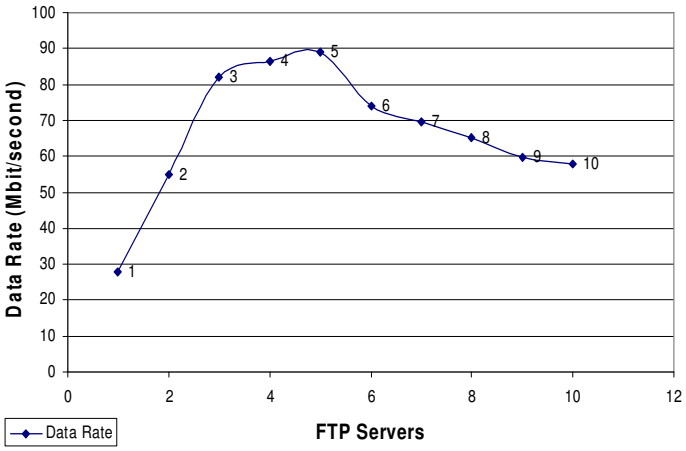

Figure 6. Using multiple parallel streams vs. single stream.

6. References

[1] Michael Cox, David Ellsworth, "Application-controlled demand for out-of-core visualization". *Proceedings of the IEEE Visualization '97.*

[2] J. Semke, J. Mahdavi, M. Mathis, "Automatic TCP Buffer Tuning" *Computer Communication Review, ACM SIGCOMM, volume 28, number 4, Oct. 1998.*

[3] Gwang S. Jung, Q. Malluhi, et al. "An Automatically Reconfigurable Distributed Data Storage System for High Data Availability". *Proceedings of the IASTEDInternational Conference Parallel and Distributed Computing Systems, 1999.*

[4]Q. Malluhi, Zeyad Ali, "DTViewer: A High Performance Distributed Terrain Image Viewer with Reliable Data Delivery". *Proceedings of the 2nd International Workshop on Intelligent Multimedia Computing and Networking IMMCN 2002.*

[5] T. Anderson, "The Case for Application-Specific Operating Systems," *Proceedings of the Third Workshop on Workstation Operating Systems, April 1992, pp. 92-94.*

[6] John Kubiatowicz, et al. "OceanStore: An Architecture for Global-Scale Persistent Storage ". *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), November 2000.*

[7] Richard W. Watson, Robert A. Coyneet al. "The Parallel I/O architecture of the High-Performance Storage System (HPSS)". *Proceedings of the 14th IEEE Symposium on Mass Storage Systems.*

[8] Tierney, B. Lee, J., Crowley, B., Holding, M., Hylton, J., Drake, F., "A Network-Aware Distributed Storage Cache for Data Intensive Environments", *Proceedings of IEEE High Performance Distributed Computing conference (HPDC-8), August 1999, LBNL-42896.*

[9] Q. Malluhi and Omar Aldaoud, "Architecture of an Efficient, Reliable, Cost-effective, Scalable, Self-healing VoD system". *Proceedings of the International Conferences on Parallel and Distributed Techniques and Applications. PDPTA'02.*