Analysis of NDN Repository Architecture and its Improvement for I/O Intensive Applications

Inchan Hwang[1], Dabin Kim[2], Young-Bae Ko[2]

[1]*Graduate School of Software Engineering,*
[2] *Graduate School of Computer Engineering*
Ajou University, Suwon, South Korea
inchan@uns.ajou.ac.kr, dabin912@uns.ajou.ac.kr, youngko@ajou.ac.kr

Abstract— **Named data networking (NDN) has been researched to be a substitute of the conventional IPv4 Internet. Its advantages are strong security, in-network caching, and mobility support. In terms of network architecture, it emphasizes contents rather than its locations to access data after the consumer's interest is routed at each node. In contrast to many NDN researches on routing, node architecture, caching, security, and other research domains, there has been a little effort on the design and architecture development of NDN based repository and its file system to save contents efficiently. So far, there are two NDN repositories, such as NDNFS and Repo-ng. They are designed to store files in NDN way so that data is sent and received over NDN network more efficiently. However, its biggest problem is the overhead when they run at I/O intensive environment, such as serving science-modelling data, which is large, sometimes over a few gigabytes. In the performance evaluation of this paper, the extent of its severity is depicted. Moreover, the investigations on those architectures will help one notice the source of the overhead. According to our findings, an enhanced metadata scheme and its management strategy will be put forward to alleviate this problem. Furthermore, some of the implementation considerations for the NDN based distributed file system for I/O intensive environment will be discussed.**

Keywords— Named Data Networking; NDN; repository; file system; future Internet;

I. Introduction

NDN[2] is a future Internet architecture. It moved from the traditional networking concept of the end-to-end connection to the data packet transmission from one hop to another hop. This new idea of networking promises better security, delay-tolerant style of communications, hop-by-hop routing, and better support of mobility. Most of all, it suggests a solution to traffic redundancy of the conventional IPv4 Internet because its traffic has dramatically increased for past years [1] since the dawn of IoT, and big data management. Its growth is unprecedented, all those data from various kinds of machines and mobile devices being connected to the Internet, has already reached its bandwidth limit. However, its integrated caches into the network architecture are expected to cut back on traffic congestion around the data producer and the network itself. Since its emphasis is on efficient content distribution, NDN researchers have devised several schemes, such as NDNFS [6], and Repo-ng [7] to optimize file transfer between nodes. Their structures are the primitive forms of serving contents to make them ready for NDN data transmission as soon as any Interest playing a role of a data seeker and retriever arrives.

Because of the new networking components of NDN which has been unconsidered in the IP based network, it triggered several potential optimizations of NDN architecture in the distributed file system. NDSS [5] integrates network and local storage data descriptions with named data. It forwards Interest based on not only Face list but also storage information so that it reduces the overhead in data format transition and suggests a new method for data operations over the distributed file system. However, it does not address performance issues when the contents are written in the file system of the OS in NDN ready mode. In addition, cached contents in each NDN node must be up-to-dated and be exact copies of the originals. FileSync/NDN [4] is an application that employed CCNx-synchronization protocol to resolve this issue.

However, due to its relatively short history of NDN compared with the conventional IP network architecture; there is still limited knowledge of its application to the NDN based repository, which contains files. Especially, when large files are inserted into the NDN repository, it shows a significant performance degradation. This problem is highly likely to occur when NDN is applied to high performance computing and Big Data fields. Two NDN storage solutions have been suggested so far, such as NDNFS, and Repo-ng have this issue although its extents are varied. We have already addressed these issues in the previous work [3] although the previous poster only points out the significance of the issues. In this paper, we also provide the solution for better performance based on architecture analysis and the observed figures. Overall, following contributions are made in this work.

- Full descriptions of current NDN repository architectures and its problems in I/O intensive environment.
- Enhanced metadata configurations and its management scheme for I/O intensive environment
- Design considerations for NDN distributed file system and its strategy to apply the enhanced scheme

First, we are going to study whether current repositories preform reasonably at serving large data files such as I/O intensive environments [8][13]. For example, science-modelling files are large and take a long time of write and read operations. In the suggestion of the improved scheme, the details of the enhanced architecture will be explained. Moreover, we provide the comparisons of those repositories and the performance descriptions of the improved scheme. Furthermore, in the last step, we outline a few considerations on its application to the development of NDN DFS, which is expected to show higher performance and require less maintenance effort.

II. Brief Overview of NDN Repositories

A. NDNFS

NDNFS is a NDN-friendly file system, a tool that helps content owners publish contents over the NDN network. It is built upon Filesystem in User Space (FUSE) [9]. Without much effort to integrate it into the native OS file system, it allows users to mount NDN file system with just few commands. When users copy and paste the file from the source location into NDNFS file system to publish it, NDNFS chops it into many NDN segments. They are in the network-ready state after the segmentation process into the file system so that when an interest for a particular data gets in contact with NDNFS daemon, it is ready to be sent over NDN. It also provides a simple working example scenario over Web with Fire-fox and NDN.JS add-on [10]. Contents published by NDNFS over NDN network can be viewed and searched with Firefox web browser. In addition, all the files and directories in the file system can be viewed and searched in the terminal and the file manager of the OS even though they are actually split in NDN packets. Other average file operations, such as read, write, modify, and delete can be done over them. Features of NDNFS are summarized below:

- *FUSE file system*: Data is written in FUSE file system so that average OS file operations like "cp" and "mv" could be performed.
- *Metadata management*: The metadata of each segment is kept in SQLite across three tables.
- *Extra tools*: It works with NDN.JS Firefox add-on so that users navigate the file system and download the published contents.

B. Repo-ng

Repo-ng is an application of NDN persistent in-network storage complying with repo-protocol allowing data management operations of Repo-ng, including file read, insertion, and deletion. Compared with the native Interest packet, Repo-ng employed the concept of signed Interest [11], modified Interest packet for the authentication of a data consumer's control command. Users insert contents at the repo with signed Interests that have encoded users' commands to do the requested operations at Repo-ng. Contrary to NDNFS, all the contents are stored in SQLite with names, and data itself. It also provides a few simple tools to insert and retrieve contents from Repo-ng. "*ndngetfile*" and "*ndnputfile*" are the ones included in its package for insertion and retrieval operations respectively. Features of Repo-ng are summarized below:

- *Data management:* Data and its name are stored in SQLite blob data type.
- *Signed Interests*: Write, read, and delete operations are done by receiving signed command Interests. This one is an extended version of the standard one, adding security features on it, authorizing the operating personnel.
- *Extra tool*: "ndnputfile" and "ndngetfile" are provided with Repo-ng to retrieve and write data in repo.

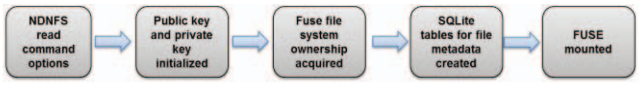
NDNFS read command options
Public key and private key initialized
Fuse file system ownership acquired
SQLite tables for file metadata created
FUSE mounted

Figure 1. Initialization process of NDNFS.

III. Architecture Analysis and Comparison

A. Initialization process of NDNFS

1) NDNFS

NDNFS undergoes the initialization process before any data is inserted. In this phase of process shown in Figure 1, it sets up all the components required to operate themselves over the network, such as keychain, database table creation, and file system initialization. The investigation of this process is the first step to learn about the blueprint of its entire architecture. First, it reads options after ndnfs execution file for configuration, such as actual directory to store files, the mount point where users access the file system, the prefix name to publish the contents over NDN, the path of the log file, and the location of the metadata DB. Afterwards, it initializes key chain to sign packets later. Then, it acquires the current user and group IDs for ndnfs access privilege. Third, it creates SQLite tables, such as "**file_system**," "**file_version**," and "**file_segments**". File system contains a list of files in the file system, its current version, its mime_type, the signature-encoding scheme [14] in the column "*type*." Another column "*ready_signed*" is unused currently but is going to support asynchronous signature, which is now being researched.

Each file's information is recorded in metadata tables, such as size of each file. Lastly, "file_segments" table is intended to accommodate the information of stored segments, for example, signature, segment sequential number, and file path. After the creation of those tables, it returns the OS with *fuse_main()* so that FUSE will work in NDNFS way with its own defined methods for Linux file operations.

2) Repo-ng

Unlike NDNFS, Repo-ng reads the configuration file to set up its environment. Then, it sets a keyboard input to terminate the application. Otherwise, it waits for command Interests to arrive at the repository daemon. Afterwards, it initializes the SQLite DB to store data and its corresponding names. It also

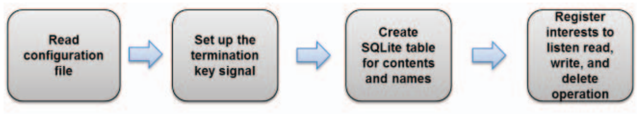
Read configuration file
Set up the termination key signal
Create SQLite table for contents and names
Register interests to listen read, write, and delete operation

Figure 2. Initialization process of Repo-ng.

saves information on public keys in "*keyLocatorHash*" column about the key selection in the public key chain so that the data consumer may decrypt the signature to verify the receiving data packets. This process is visualized in Figure 2.

B. File insertion

1) NDNFS

When a user wants a file to be published over NDN, one copies it onto the file system. When it is pasted into the mount point of the FUSE, NDNFS investigates whether there is already a file identical to it. If not, it reads the list of files from the target directory. If there is not a file to write, it creates a new file to append segments being copied from the source. Next, it makes sure the target file's existence, reads the file's extension, decides the mime type, and saves it into the SQLite database.

Finally, it begins writing to the file, the target file is incremented by 4KB, and the increment continues until it reads the end of the source. Each step, it writes version information into SQLite table. After it completely finishes writing the file, here comes the most expensive operation next, sign each segment, and store it into the SQLite. This step is indispensable. NDN specification demands all data packets to be signed for stronger security. And then, write operation is finished.

2) Repo-ng

File insertion into the Repo-ng is different from NDNFS. Instead of the utilization of OS file operation calls, repo file insertion tool like "*ndnputfile*" issues file insertion Interests to the Repo-ng. After the Interests arrive at it, it recognizes the authentication on the command Interest. Only authorized Interest insertion command allows a file to be stored into it. This Interest also has information on the "*StartBlockID"* and "*EndBlockID*" of the desired data, or it has the selector information to retrieve it.

Once Repo-ng has the enough information to retrieve the target data, it expresses Interests to the target node where the target contents are placed. It receives data packets in response to the previously issued Interests. After the last data packet landed, "*ndnputfile*" sends insertion status check Interest to the Repo-ng. If there has been no error occurred at Repo-ng, it answers with insertion complete message.

C. Observations on SQLite tables

As depicted in Table 1, Repo-ng has three blob typed attributes, and one column with integer, which functions as the primary key of the table. Data attribute reserves all data packets, and the name attribute contains its corresponding name. "*keyLocatorHash*" attribute has the information on a key to decrypt the signature in the data packet. All of these attributes' type is blob.

However, as shown in Table 2, "*path"* attribute of NDNFS that is identical to "*name*" attribute in Repo-ng is a text type. There is no blob data type in it except for "*signature*" attribute, which seem to result in higher performance than that of Repo-ng, described in next section.

TABLE 1. NDN_REPO DB SCHEMA OF REPO-NG

Attribute	**Type**	**Description**
id	integer	primary key of the table.
name	blob	NDN name for contents
data	blob	contents in binary format
keyLocatorHash	blob	public key information

TABLE 2. FILE_SEGMENTS DB SCHEMA OF NDNFS

Attribute	**Type**	**Description**
path	text	file path(equivalent to "name" in Repo-ng)
version	integer	version information of each segment
segment	integer	number for each segment
signature	blob	signature of each segment

TABLE 3. FILE_SYSTEM DB SCHEMA OF NDNFS

Attribute	**Type**	**Description**
path	text	content path("name" in Repo-ng)
current_version	integer	content version
mime_type	text	mime type of a file
ready_signed	integer	each file's signature state
type	integer	file type, the output of ls command

TABLE 4. NEW METADATA CONFIGURATION FOR EACH CONTENT

Attribute	**Type**	**Description**
path	text	content path("name" in Repo-ng)
current_version	integer	content version
mime_type	text	mime type of a file
ready_signed	integer	each file's signature state
type	integer	file type, the output of ls command
certificate	binary	producer's certificate for each content

IV. DISCUSSION AND ENHANCED SCHEME

Both NDN repositories make use of SQLite. However, their performance statistics show significantly lower performance than the average Linux file system. Their differences could be due to the uses of blob data types in SQLite tables [12]. Repo-ng's data storage algorithm totally relies on SQLite, but NDNFS also exploits FUSE along with it. For better performance of SQLite, Repo-ng stores all the data in memory first like contents, names, and the signature itself. This algorithm could hurt overall system performance when it inserts a large size file, like big data and other data intensive science files, resulting in as drawback to the application to high performance computing. Data shown in the table 3 depicts that Repo-ng's write operation takes up a large portion of the system memory due to those reasons. In addition, NDNFS also suffers performance degradation because of SQLite. When it is compared with a file operation of Ext4, severe bottleneck is observed. It is caused by frequent write operations of signature and other data into SQLite blob data type.

A. Problems of SQLite and its DB tables

SQLite was used to save the metadata of a content. According to SQLite developer's document, there are many issues regarding its use on the file system [15].What if this file system implements file locking? There is a possibility of data corruption in DB when file locking runs inaccurately. In addition, when a requirement is changed to have higher concurrency, especially in the case of multiple writings to the disk, there is overhead in I/O. NDN is supposed to work efficiently in this circumstance. Thus, SQLite is a drawback providing negative QoS to the users.

Therefore, a new NDN data-storing scheme, which makes no use of SQLite, must be suggested to apply NDN storage in the high performance-computing field. The current NDNFS makes metadata and keep them in order to serve it over the network immediately. It incurs overhead while doing file operations into the file system, such as writing, renaming, copying, modifying, and so on because these operations update all the packet information in SQLite. One may recognize the fact that many attributes belong to a file, and they are derived from it. Attributes, such as path, version, mime_type, and so on are about a file. All of them may be relocated into a file itself, not in SQLite for the sake of performance as it is considered being a drawback in an I/O intensive environment.

As previously explained, Repo-ng stores the content and its metadata into blob data types. According to the SQLite community, it only brings about extra processing time when it reads a large sized file. Table 1 shows that Repo-ng's metadata has already been designed to be a per-file structure. However, "keyLocatorHash" attribute only contains the information to decrypt signature. There is expected to be a dedicated column for an encryption scheme of the content for an enhanced scheme.

B. Metadata management with xattr

According to the performance evaluation, database utilization has only been a burden to the system. Although embedding database to the file system has traditionally been mandatory, this approach brings about a problem on performance. The evolving NDN protocol will produce a lot more metadata to save contents on the disk, the best solution is always to minimize the DB involvement into the file system. The creation of per-file metadata scheme is indispensable because it will not only reduce DB dependency, but also depicts possibility of employing other metadata management tools, such as extended file attribute (xattr) [18].

Xattr is a key-value pair database like information storage tool supported by a file system to a file. Most of modern Linux file systems support it. Except for signature, all other metadata was able to be stored in a file because signature is a per-packet metadata.

C. Avoiding Signature Write

Thus, a new metadata configuration strategy must be proposed to replace per-packet signature. As shown in Table 4, the new metadata strategy includes the certificate issued by the producer for the content. Therefore, it will remove the necessity of caching signatures but still have them because it will generate them at the network interface when the contents are consumed first time or later again. This scheme is made feasible if there is a complete trust, authentication process, between the file system (server) and the writer (data producer) so that each certificate of a file is safely stored in the file system. This scheme will eliminate all the problems listed in the previous section, such as renaming and updating contents at NDN repositories. But Repo-ng will not have such overhead. But its writing operation into SQLite blob data type will be a source of performance drop.

V. Performance Evaluation

We compared the performance of those two repositories and the prototype of enhanced NDNFS while they put files into the storages. All the experiments are carried out on an Ubuntu 15.10 system with Intel Xeon® CPU 3.4Ghz 8 Cores, 14GB RAM, 50GB hard drive. The NDN platform we used were NFD v0.4.0, NDN-cxx v0.4.0-beta2, NDNFS v0.3, NDN-CPP v0.4.0, and Repo-ng released on Nov. 21st, 2015. We selected 5 sample files weighing 200MB, 700MB, 1GB, 2GB, and 3GB respectively. We did not modify the default packet size set in NDN-cxx so that data transmission between "*ndnputfile*" and Repo-ng was facilitated in 8K. To collect benchmark datasets, a bash script, running "ps" commands per second to read memory consumption and CPU resource, was made and it wrote the readings into text files

The statistics in Table 5 were collected while several large files were inserted into Repo-ng and NDNFS. Generally, Repo-ng performs worse than NDNFS. In case of 1GB file insertion, Repo-ng took up more memory space than NDNFS. It took approximately 77% of the total system memory while NDNFS never took more than 0.1% of the total space. It also used nearly 100% of CPU resource whereas NDNFS consumed around 60% of its total. This memory consumption indicated because Repo-ng stores all the data, such as contents, names, and the signature in memory until it finishes reading. The reason for low memory consumption of NDNFS comes from its size of data writing into the file system at each cycle. It is always fixed to be 4KB, which is nearly 0% of the system memory.

TABLE 5. INSERTION PERFORMANCE COMPARISON

Metrics	200MB		700MB		1GB	
	NDNFS -port	**repo -ng**	**NDNFS -port**	**repo -ng**	**NDNFS -port**	**repo -ng**
CPU(%)	60%	100%	59.3%	100%	63%	100%
Memory	0.1%	16.3%	0.1%	55.3%	0.1%	77%

Figure 3 shows time taken values across a variety of sample big data files. In general, Repo-ng takes longer to insert a file into the repository. In case of 1GB file, Repo-ng took 2215.4 seconds in average to finish insertion. However, only 39.8s elapsed with NDNFS for the same file. This trend is repeated with other cases. 1130.4 seconds in average passed while Repo-ng was inserting 700MB, but it only consumed 22

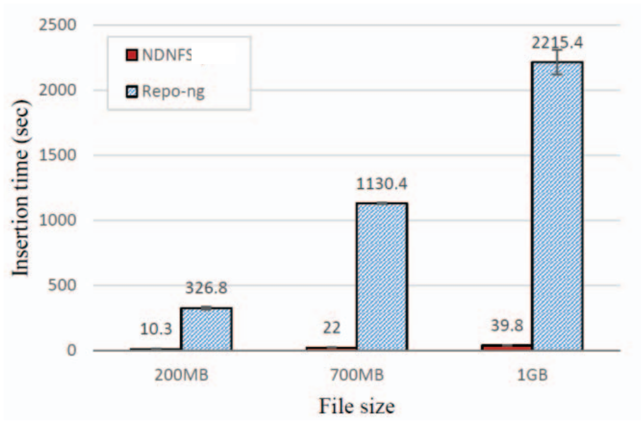

Figure 3. Comparison of insertion time

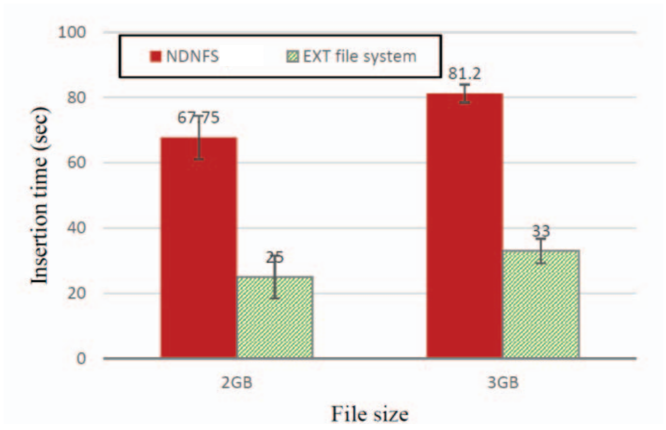

Figure 4. Comparison of insertion time

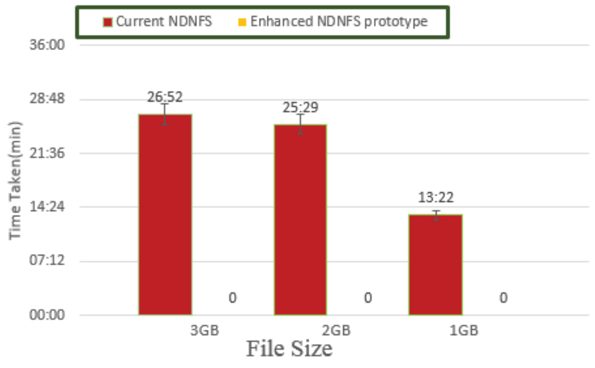

Figure 5. Time consumption for signature assignment

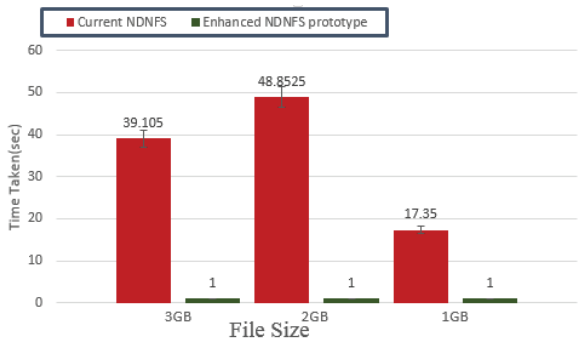

Figure 6. Time consumption for renaming operation

seconds for NDNFS. As for 3GB and 2GB files, Repo-ng was unable to complete the insertion. It kept consuming the system memory and space in the swap file. The system-monitoring tool showed that Repo-ng used up all of 14GB RAM and the virtual memory, and the system crashed for inadequate memory space. Thus, those values were detached from the table because it was unable to record the values for those 2GB and 3GB files. Time consumption is due to its one-to-one NDN communication nature. "*ndnputfile*" and Repo-ng exchange interest and data one-by-one in NDN manner. It is an extremely slow process.

Figure 4 is a graph of time measurements while 2GB and 3GB files were being inserted into a destination folder and the NDNFS. Although file operations of NDNFS works quicker than the other one, It still operates slowly in comparison with ordinary Linux file system, Ext4. This bottleneck is believed to be caused by SQLite operations' frequent SQLite operations per each disk operation cycle [8]. Although it took 81.2 seconds for 3GB sample file to finish "cp" command for insertion, there was an additional step to write signatures in the release operation in the FUSE afterward.

As depicted in Figure 5, it spent 26:52 minutes of extra time, approximately 30 min to complete a writing operation to the file system. During the release operation, the user may begin another write. However, the performance significantly dropped due to the signature write being done at SQLite.

Figure 5 also shows that different lengths of signature writing operations are run in the background as to their file sizes. It hurts the performance of other file operations during the SQLite task. However, our prototype implementation with the proposed scheme does not undergo this step, resulting in better performance for I/O intensive environment overall.

Figure 6 presents that renaming a file also takes a significant amount of time in the current scheme because it has to rename all the signatures and other metadata in SQLite. As for the 3G sample file, it took 39.105s to rename it. However, our prototype NDNFS with improved scheme for I/O intensive system does not make use of SQLite nor record signatures into it. It finishes renaming instantly taking around 1 second for all sizes of the contents.

VI. CONSIDERATIONS ON ITS EVOLUTION TO DFS

According to Satyanarayanan's work [19], distributed file systems employed client/server database to maintain metadata. As for I/O intensive system, database could cost so much depending on its I/O intensity. To achieve its reliability and throughput, a reliable product must be integrated into the file system, such as Oracle DB, costing extra license fees as to the number of CPU cores and accessed users. However, its cheaper replacement, MySQL has a less reliability than Oracle product [16][17]. If a database embedding into a distributed file system is a crucial element, sharing its load with

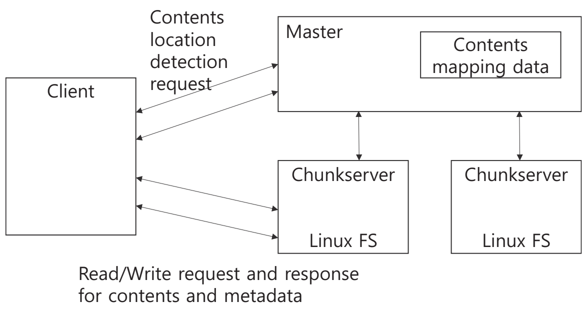
Contents location detection request
Master
Contents mapping data
Client
Chunkserver
Linux FS
Chunkserver
Linux FS
Read/Write request and response for contents and metadata

Figure 7. Conceptual diagram of NDN DFS

alternative means of database, such as xattr may take less cost and provide better performance. Further research on the application of NDN based distributed file system metadata configuration in consideration of xattr alternative will offer advantages on NDN applied system. Brief philosophy of the NDN DFS metadata design is followed below.

- *Central Metadata Server*: Central metadata server, probably operated by traditional client/server database, has a minimal dataset to map the locations of the entire contents across DFS.
- *Each Content File*: All the other metadata derived out of each file is recorded into itself. For example, in the form of the extended file attribute.

The central server role must be limited to map the DFS members and its contents' locations, and all the other extra information must belong to its own file as portrayed in Figure 7. In this way, the throughput increases and its overhead decreases due to the minimal workload on the metadata server.

VII. Conclusions

We have concluded that the application of SQLite into a NDN file system is inappropriate for I/O intensive environment although this approach is in the notion of the traditional DFS to manage metadata. However, this approach aroused overhead in the current NDN file system at I/O intensive environment. Thus, we employed an alternative approach with a new metadata configuration to replace SQLite, reducing overhead in the file system. We also put forward to a design suggestion to minimize DB involvement for NDN based DFS based on our findings, reducing maintenance cost as well as performance improvement. Our future work will be researching on quick signature generation algorithm in cooperation with a graphic processing unit to reduce overhead caused by it, and the full design and implementation of DFS with our enhanced prototype of NDNFS.

Acknowledgment

This research was supported by the MISP(Ministry of Science, ICT & Future Planning), Korea, under the National Program for Excellence in SW) supervised by the IITP(Institute for Information & communications Technology Promotion)(R22151610020001002)

References

[1] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020 White Paper", Cisco, 2016. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in ACM CoNEXT, Rome, Italy, 2009

[3] I. C. Hwang, D. Kim, Y. B. Ko, "Analysis of NDN Repository Architectures" in IEEE International Conference on High Performance Computing(HPCS 2016) , Innsbruck, Austria, 2016

[4] J. Lindblom, M. C. Huang, J. Burke, L. Zhang, "FileSync/NDN: Peer-to-Peer File Sync over Named Data Networking," NDN Technical Report NDN-0012, March 2013

[5] S. Chen, J. Cao, L. Zhu, "NDSS: A Named Data Storage System," in IEEE International Conference on Cloud and Autonomic Computing (ICCAC), Cambridge, MA, USA, 2015

[6] W. Shang, Z. Wen, Q. Ding, A. Afanasyev, and L. Zhang, "Ndnfs: An ndn-friendly file system," NDN Technical Report NDN-0027, Revision 1, Oct. 2014

[7] Repo-ng: Next generation of NDN repository, *http://redmine.named-data.net/projects/Repo-ng/wiki*

[8] C. Fan, S. Shannigrahi, S. DiBenedetto, C. Olschanowsky, C. Papadopoulos, and H. Newman, "Managing scientific data with named data networking," in ACM SIGHPC Workshop on Network-aware Data Management (NDM), Austin, Texas, 2015

[9] FUSE: Filesystem in Userspace. *http://fuse.sourceforge.net/*

[10] W. Shang, J. Thompson, M. Cherkaoui, J. Burke, and L. Zhang. "NDN.JS: A javascript client library for named data networking," in IEEE INFOCOM Workshop on NOMEN, Turin, Italy, 2013

[11] "Signed Interest." [Online]. Available: http://redmine.named-data.net/projects/ndn-cxx/wiki/SignedInterest

[12] Internal Versus External BLOBs in SQLite. https://www.sqlite.org/intern-v-extern-blob.html

[13] W. Yu, S. Liang, D. K. Panda, "High performance support of parallel virtual file system (PVFS2) over Quadrics," Proceedings of the 19th annual international conference on Supercomputing, Boston, MA, 2005. DOI = http://dx.doi.org/10.1145/1088149.1088192

[14] D. K. Smetters and V. Jacobson. Securing network content, October 2009. PARC Technical Report

[15] SQLite Document, "Appropriate Uses for SQLite," [Online]. Available: https://www.sqlite.org/whentouse.html

[16] DB Engines, "System Properties Comparison Microsoft SQL Server vs. MySQL vs. Oracle," [Online]. Available: http://db-engines.com/

[17] ITX Design, "MySQL vs Oracle," [Online]. Available: https://itxdesign.com/mysql-vs-oracle/

[18] Linux Programmer's Manual (2015) xattr-Extended attributes [Online]. Available: http://man7.org/linux/man-pages/man5/attr.5.html.

[19] M. Satyanarayanan, "A Survey of Distributed File Systems," Technical Report CMU-CS-89- 116, Department of Computer Science, Carnegie Mellon University, 1989

In Chan Hwang earned his BS degree in Computer Science from Georgia Southwestern State University, GA, USA, in 2013. He is currently attending the graduate school of Software Engineering at Ajou University. His research interest is in the area of Named Data Networking.

Dabin Kim received her BS degree in Computer Science from Seoul Women's University, Korea, in 2010. She earned Ph.D. at the school of Computer engineering of Ajou University, Korea. Her research interests are in the area of wireless communication and information-centric networking.

Young-bae Ko is currently a Professor in the Department of Software at Ajou University, Korea, leading the Ubiquitous Networked Systems Lab funded by BK21+ (Brain Korea 21 Plus) National Project. Prior to joining Ajou University in 2002, he was with the IBM T.J. Watson Research Center, New York, as a research staff member in the Department of Ubiquitous Networking and Security. He received his Ph.D. degree in computer science from Texas A&M University. His main research areas are in wireless ad hoc/mesh networks, IoT, Smart Grid and Future Internet. He was the recipient of a Best Paper award from ACM Mobicom 1998. He has served in various activities, most notably as the general chair in IEEE SECON 2012 and editorial board of Elsevier Digital Communications and Networks. See http://uns.ajou.ac.kr for further details.