Cache Management for Large Data Transfers in Named Data Networking using SDN

Mohammad Alhowaidi, Deepak Nadig and Byrav Ramamurthy
Dept. of Computer Science & Engg, University of Nebraska-Lincoln, Lincoln, NE 68588-0115, USA.
Email: {malhowai,deepaknadig,byrav}@cse.unl.edu

***Abstract*—The Compact Muon Solenoid (CMS) on the Large Hadron Collider (LHC) manage high volumes of data that currently exceeds 100PB across different sites. An important challenge of delivering data to experimenters in the CMS workflow is the data volume. An experiment data file has an average size of 2 Gigabytes, with file sizes ranging between 100 Megabytes and 20 Gigabytes. Also, a complete dataset comprises of multiple files, with the dataset files ranging from 2 Terabytes and 100 Terabytes in size. Providing fast access to datasets is an important enabler for data-intensive science research. In our work, we demonstrate a Information-Centric Networking (ICN) approach to providing fast in-network access to CMS datasets. To that end, we must first address the problem of how to store large CMS files in network caches closer to the end-users. We propose a software-defined, storage-aware routing mechanism using named data networking (NDN) to achieve this goal. Due to the inherent capacity limitations of the NDN router caches, we use software defined networking (SDN) to provide an intelligent and efficient solution for data distribution and routing across multiple NDN router caches. We demonstrate how software-defined control can be used for partitioning and distributing large CMS files based on NDN router cache-state knowledge. Further, SDN control will also configure the router forwarding strategy to retrieve CMS data from the network. Using our proposed architecture, we show that CMS dataset can be retrieved 28% – 38% faster from the NDN routers caches compared to existing approaches. Lastly, we develop a prefetching mechanism to improve the transfer performance of files not available in the router's cache.**

I. Introduction

Numerous scientific experiments rely on large data transfers associated with data-intensive science. An example workflow is the Compact Muon Solenoid (CMS) experiment. The CMS experiment [1] on the Large Hadron Collider (LHC) manages a large volume of data that currently exceeds 100PB across multiple sites. The experiment manages approximately 35PB of data (a combination of detector readouts and simulated readouts across a variety of physics-related formats); this data is write-once, read-many. All CMS managed data is immutable once written to the permanent storage [2]. Through a combination of caching and pre-placement, CMS moves its data across 50 data centers throughout the Worldwide LHC Computing Grid [3].

The main challenge of delivering data for CMS experimental workflows is the large size of the dataset. CMS experimental dataset sizes vary between 2 to 100 Terabytes. In this work, we study how to leverage Information-Centric Networking (ICN) to provide faster in-network CMS data access to end-users. Contrary to IP-based, host-centric Internet architectures, ICN emphasizes content by making it directly addressable and routable. The users request the data based on its name instead of using IP addresses. Named Data Networking (NDN) [4] is an example of an ICN architecture that transfers data by sending *Interests packet* and receiving *Data packets*. An essential feature of NDN design is the use of an in-network cache on the routers. These storage-enabled routers are placed between content custodians (origin servers) and users. The main advantage of serving the data to the user from these routers is that the retrieval latency will be reduced and the user will get the data faster.

A critical problem that must be addressed to use NDN with CMS workflows is how to efficiently store very large files in the network. The current implementation of the NDN content store (CS) is based on caching the data *in-memory* to provide faster data access. Due to the limitations of the NDN cache capacity on each router, novel approaches for efficient cache management are necessary. For instance, NDN routers can be deployed with large memory, however, this will increase the deployment costs and is therefore inefficient. Another approach is to use a solid-state drive (SSD) for caching the data, but this approach will increase both the cost, and add additional delays for data retrieval. In this work, our goal is to find the appropriate mechanisms to create a software-based, cache-aware router based on centralized controller and NDN. Using a centralized controller like the SDN controller [5] is an important area of interest for the Information-Centric Networking (ICN) community. The SDN controller can provide intelligence for routing and caching management by decoupling the ICN data plane from its control plane.

Our proposed solution works in two phases. First, during the file retrieval process, if the file is not cached in the network (and resides on the producer storage), the interest packet will be forwarded to the centralized controller for the best retrieval strategy. Small files are retrieved using the default NDN approach. However, for large files that cannot be cached on a single router, a distributed retrieval approach using multiple router content stores will be used. Second, depending on whether the requested file is already cached on multiple routers on not, the controller will provide a strategy for distributed file retrieval (See Section III). On the other hand, our system architecture also enables the *prefetch feature*, where parts of the file can be prefetched and cached on different routers simultaneously.

The paper is organized as follows: Section II presents the NDN background and the related works; In Section III, we describe our system architecture for SDN control of NDN; Section IV outlines our solution approach for cache management. We describe our testbed setup and discuss the results in Section V. Finally, we conclude our work in Section VI.

II. Background and Related Work

A. Named Data Networking

NDN is a Future Internet Architecture (FIA) project which proposes to re-design the current host-centric Internet architecture. NDN requests data using its name instead of the IP address. There is two types of packets in NDN: (i) Interest packet which contains the data name and is sent to retrieve the data, and (ii) Data packet containing the data to be sent back to the consumer as a response to the Interest packet. NDN names are hierarchically structured. For instance, the name */ndn/repository/file*, is carried by the Interest and is used to forward the data to the content custodian.

NDN Forwarding Daemon (NFD) is responsible for routing the interest and caching the data. NFD manages three data structures: Pending Interest Table (PIT), Content Store (CS), and Forwarding Interest Base (FIB). The NDN consumer generates and sends the Interest packet in the NDN network. When the router receives the Interest packet, it uses the data name to forward the packets to the NDN producer (i.e., data custodian). The router will store the Interest in its PIT along with the incoming interfaces. If another Interest, from a different consumer, requesting the same data reach the router, then the PIT will return the Data packet into the consumer upon receiving it. The retrieved data is cached in the routers' CS; if the data is cached in the router CS, then the router will reply directly to the consumer without sending the Interest packet to the producer. The caching process is an efficient way to reduce the latency in retrieving the data and reduce overheads on the producer. The FIB is considered as the routing table for the router.

An important feature that NDN routers have is the forwarding strategy that is used to define how and where the packets are forwarded. The forwarding strategy will choose a specific next hop for forwarding until the Interest reaches the destination. Also, it can select different paths to retrieve the data. For instance, the forwarding strategy can choose to forward the packet on the shortest (number of hops) path, the lowest latency path, or the least congested path.

B. Related work

Numerous recent works explore the use of centralized control for managing information centric networks (ICN). Works such as [6], [7], [8], and [9] explore cache placement, caching policies, and content selection strategies on both on- and off-path routers. SDN-based control of ICN is also proposed for distributed data transfers in data-intensive science [10], [11], [12]. The authors in [13] discuss multi-path interest distribution strategies for both distributed and centralized control of NDN and how it can improve data transfer performance. However, the interest distribution strategies presented in [13] do not consider network layer properties such as NDN on-path congestion and the route bandwidth availability. Different from the above works, we propose a centralized cache management framework, that consider the limitation of the NDN router cache for large data transfer. Previous works deal either with data retrieval performance and cache placement, or develop techniques for off-path data retrieval. However, the missing piece in these works, which we address in this work, is the cache management framework and the collaboration between different routers' CS to cache large files and return it to the user efficiently.

III. System Architecture

NFD forwarding strategy is responsible for choosing the interest forwarding interface. The interest can follow a different route based on the forwarding strategy used. Designing and choosing the correct strategy will affect the performance and the data retrieval process. Several forwarding strategies are provided by the NFD such as best routes, multicast, access router, client control, and adaptive SRTT-based Forwarding (ASF) [14] strategy.

The above strategies can be used in different network environments but they are unsuitable for large-scale datasets. Since NDN relies mainly on caching the data in the router to reduce retrieval latency, the above strategies ignore the environments that deal with large datasets. In this paper, we propose a solution for cache management and develop a forwarding strategy that deals with large datasets. We develop an architecture and associated strategies to deal with large-volume distributed data transfers over high-bandwidth, high-delay wide area networks (WANs). The targeted use of our architecture is the complex and distributed filesystems such as CernVM File System (CVMFS) [15]. High-energy physics (HEP) workflows (e.g., Compact Muon Solenoid (CMS) [1]) are evaluating the use of CVMFS for the distribution of experimental datasets [10] using NDN.

A. Explanatory Example

For large data transfers that exceed the Router CS size, a cache miss will always occur and the data will always be fetched from the producer. The following example illustrates the need for our proposed solution:

If the NDN CS size is 500MB (default size in the current NDN implementation) and we have a 1GB file with name *"large"*. The file will be segmented into several chunks and fetch by several interest packets. The interest name will be, for instance, */ndn/large/segNum=1* for the file's first chunk. For simplicity, let us assume that the chunk size is 1MB. Then, we will have 1000 interests to fetch the file *"large"*. Since the CS size is 500MB then the chunks 1-500 will fill the CS, then the chunks 501-1000 will start replacing the chunks in the CS if a regular replacement strategy (such as FIFO or LRU) is employed. Now, if the file is requested again, by the same consumer or another consumer, then the interests will start by fetching the first chunk. Since the CS currently contains chunks (501-1000), then a cache miss will occur, and the interest will be forwarded to the producer. The chunks (1-500) will again fill the CS and result in a cache miss. This scenario will be repeated when fetching the chunks (501-1000) be replacing the previous chunks.

B. System Architecture

Figure 1 shows an overview of our system architecture. All routers and the producer communicates with a centralized controller for cache management and data retrieval. The centralized controller manages all routers' CS and the forwarding strategies that need to be installed on the routers. The system architecture is described below:

1) NDN Consumer: The NDN consumer represents the user requesting the data. The Consumer in our architecture is unaware of the underlying architecture and requests the data directly by sending Interest to the network.

2) NDN Router: The NDN routers are responsible for caching the data in their own CS. Also, the NDN routers will use the forwarding strategy to forward the interest to the next-hop router. The NDN routers collaborate with the controller while storing the data. If the file size is small, then one NDN router is used to store the file. However, for large files, the file will be stored on multiple NDN routers' CS. Splitting the file among multiple NDN routers will avoid premature CS cache replacement. It will also ensure faster data retrievals due to the file transferred from in-network caching.

3) NDN Producer: The NDN producer represents the content custodian or the storage where all data resides and caters to the consumer interests/requests. The producer can store the actual files or store them as NDN packets. Storing the data as NDN packets in advance will avoid the overhead from converting the regular files into NDN signed packet. In this work, we convert and segment the files into NDN packets. For segmenting the file we retain the default NDN chunk size (i.e. 8800 bytes). The NDN producer will update the centralized controller about the files that it has in its repository and the size (number of chunks) of each file. The controller will store this information in its database for the cache management purposes.

4) Centralized Controller: The controller in this work represents the main entity in the managing the interest routing and caching. The controller will receive the interest from the routers. Based on the status of each router's CS and the file size, it will send the forwarding strategies for the corresponding routers to cache the whole file or part of the file in their CSs. The centralized controller uses representational state transfer (REST) application programming interfaces (APIs) for communication with the NDN routers and the NDN producer. The controller manages the NDN network state information including routers CS states (i.e. the available space on CS), forwarding paths, and data information on the NDN producer. We run a program alongside the NFD on all NDN routers and the NDN producer. This program is responsible for internal communication with the NFD and external communication with the SDN controller. The controller asynchronously communicates with this program to create a data map of all routers' CS.

Figure 2 shows an example of the use of our architecture. When consumer C want to retrieve a large file from the NDN producer, it will create an interest and send it to router R1. If it is a new interest, then router R1 will send the interest to the controller. The controller will read the file information and the Router CS from its database. The controller finds, in our example, that routers R2, R3, and R7 have enough space in their CS to store the large file. The controller will send the configuration to router R1 to split the interest for the file segments to three paths. If the file is split into n segments ($S_0 ... S_k .. S_r .. S_{n-1}$), these segments will be retrieved as follows:

- $S_0 - S_{(k-1)}$ through path $R1 \rightarrow R2 \rightarrow R5 \rightarrow P$
- $S_{(k)} - S_{(r-1)}$ through path $R1 \rightarrow R3 \rightarrow R6 \rightarrow P$
- $S_{(r)} - S_{(n-1)}$ through path $R1 \rightarrow R4 \rightarrow R7 \rightarrow P$

The controller will configure the routers R2, R3, and R7 to cache these segments. Simultaneously, the controller configures the routers R4, R5, and R6 not to cache those segments. This cache management strategy will help in reducing the number of cache misses on other routers. Further, it also benefits from other routers in caching other files.

C. Prefetch Files Segments

We developed a prefetch mechanism to reduce the latency in retrieving the files. In our previous example (Figure 2), the controller will ask routers R3 and R7 to start a parallel data segment retrieval. Since R1 will start retrieving segments ($S_0 - S_{(k-1)}$)) first. The controller will tells router R3 and router R7 to retrieve segments $S_{(k)} - S_{(r-1)}$) and segments $S_{(r)} - S_{(n-1)}$), respectively and store them in their CS. So, when R1 completes retrieving the first set of segments from the producer, it will start retrieving the second and third set of segments from routers R3 and R7 instead of the producer. In this way, the path latency will be reduced and the retrieval time will be optimized.

IV. Solution approach

In this section, we describe the implementation approach for cache management and data distribution.

A. Controller algorithm

When the router receives an Interest, it will forward it into the controller asking for the best configuration to forward the Interest. Algorithm 1 shows the process on the controller to build the configuration file. The controller will take the Interest as input. The controller will receive the file information and the routers' CS status from its data map (database). The controller will sort the routers based on the available space in each router's CS. The routers with the largest CS available space will be used first. Based on the file size and the available CS space of each router, the controller will decide the number of routers needed to cache the file. Sorting the routers based on the CS space will avoid splitting the

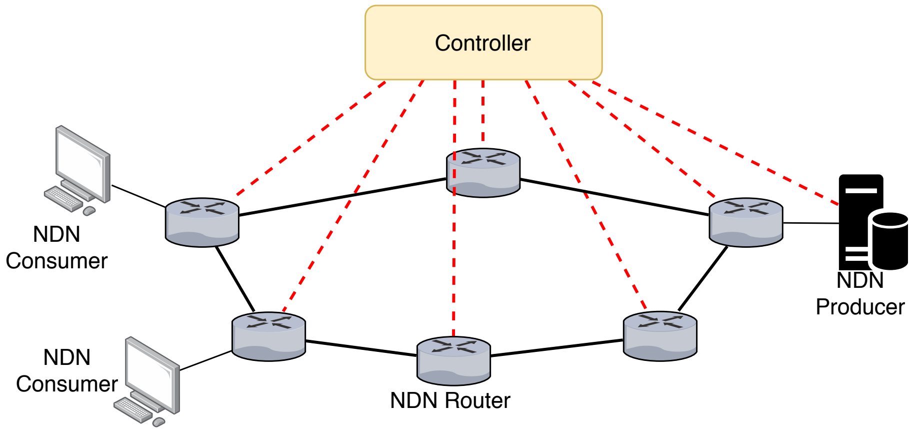


Fig. 1: System architecture.

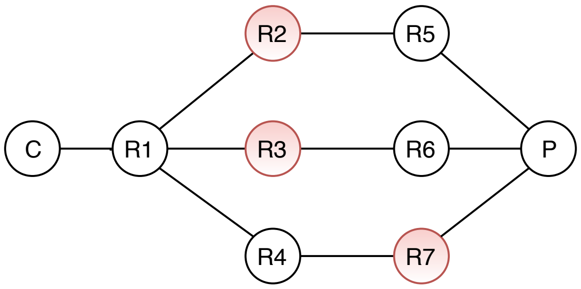


Fig. 2: Distributed Interest example.

file among a larger number of routers and avoid additional delays in retrieving the file later. The controller will send the configuration instructions to the routers on how to forward Interests for the file retrieval.

Algorithm 1 Controller Config(i)

```
Input: NDN Interest ($i$).
Output: SD-NFD Configuration File.
1: Lookup data map for the namespace in Interest $i$
2: Sort the routers based on the available space of the routers CS
3: Compute the number of routers that is needed to cache the file
4: return NFD-Config($i$)
```

B. Router Forwarding Strategy Algorithm

Algorithm 2 explains the procedure on the router. We develop a program for communication between the NFD on the router and the SDN controller. When an interest reaches the router, the router will check if it is requesting a chunk from the file which has an installed configuration, or if this interest is a new one. If the interest is new, then the router will send the Interest information to the controller. The controller will reply with the instructions, on how to forward the Interest, carried in NFD-Config file. The router will use this file to forward the Interests to retrieve the specific data accordingly. The NFD-Config file carries the information on how to divide the Interests by requesting the file from several interfaces. Since the file is segmented into several chunks, each Interest will retrieve a specific chunk. The NFD-Config file will tell the router to forward a set of Interests requesting $(segment_0, segment_{i-1})$ on one interface, and another set of Interests requesting $(segment_i, segment_{j-1})$ on a different interface.

Algorithm 2 Router configuration(i)

```
Input: NDN Interest ($i$).
Output: Router NFD configuration for data transfer.
   Configuration request:
1: if Interest not configured then
2:     Send Interest information to the controller
3: else
4:     Forward Interest based on the existing configuration
5: end if
   Configuration arrival:
6: Read NFD-Config($i$)
7: for all faces in NFD-Config do
8:     Forward Interests of $(segment_i, segment_k)$
9: end for
```

On the other hand, the controller will send instructions to the corresponding routers that are going to store the file segments in their CS to enable the caching process for these file segments. If the router does not receive this message, then it will just pass the data without caching it.

Algorithm 3 Prefetch procedure

```
Input: prefetch_MSG.
Output: Request and cache file segments.
1: READ prefetch_MSG
2: for Seg = startSeg; Seg < endSeg; Seg++ do
3:     Interest = /ndn/fileName/Seg
4:     Send Interest
5: end for
```

Further, in the prefetch scenario, the controller will inform the NDN routers that are going to cache the file to start fetching parts of the file. The first router will retrieve the file segments in the regular way (Interests coming from the consumer), while all other routers which cache the other parts of the file will start issuing Interests to store the corresponding segments in their CS. Algorithm 3 shows our prefetch process. Once the router receives the prefetch_MSG from the controller, the router will read the file name, the prefix, and the range of the segments that are needed to be fetched. Then, the router will issue these interests and cache the segments in its CS even before the actual Interests sent from the NDN consumer.

V. Results and Discussion

In this section, we explain our testbed setup, our experiments on this testbed, and discuss our results.

A. Testbed

Figure 3 shows our network topology that we used to run our experiments. The testbed consists of one NDN consumer, four NDN routers, an NDN producer, and a controller node. The NDN Consumer, all NDN Routers, and the NDN Producer are running *NDN-cxx* and *NFD*.

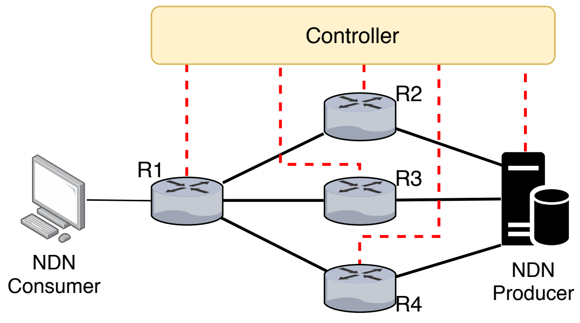


Fig. 3: Network topology.

We used the GENI [16] platform as our network testbed. GENI provides a platform for at-scale networking research, connecting compute resources over the Internet2 AL2S infrastructure. We use a total of seven GENI sites (with one node per site) spread across InstaGENI infrastructures at Kentucky MCV, Kentucky PKS2, Clemson, Texas, Wisconsin, Vermont, and Hawaii. Therefore, this setup is representative of a real-world WAN NDN network.

B. Results and Discussion

The WAN data transfer performance of the proposed architecture was tested on the GENI network platform. All nodes (i.e., controller, consumer, routers, and producer) were placed on different InstaGENI sites and aggregated using layer-2 stitching over Internet2 AL2S. Three different files (600MB, 800MB, 1GB) were used to compare the performance between the default NDN and our proposed architecture. For all files, we evaluate the transfer performance when i) The requested data is not available in the routers' CS, and the requested data is always fetched from the producer and then cached at the router(s); and ii) The file request occurred after the previous step, i.e., the data might be available in the Router CS since a similar request has been executed earlier. We computed the results with 95% confidence interval.

In all experiments, we used the default NDN router CS (500MB). The replacement policy for the routers CS is least recently used (LRU). The files are segmented and converted into NDN packets. Each segment uses the default NDN segment size (8800 Byte). We set the interest pipeline depth to 50; the consumer will send 50 Interest simultaneously before receiving the corresponding data. We set a static value for the pipeline depth to increase the transfer performance. This value will be changed into a dynamic value based on the network/path conditions in the future work.

TABLE I: Experiment type.

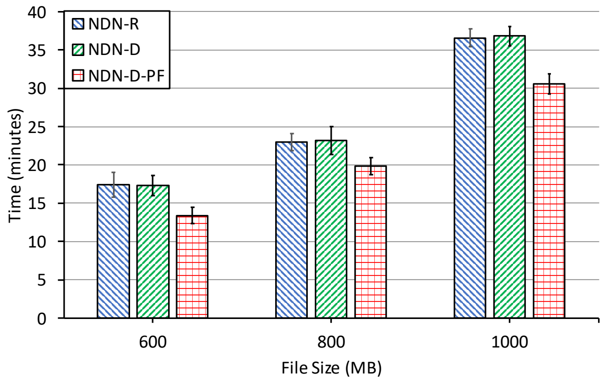
Experiment type	File location
NDN-R	Default NDN architecture
NDN-D	Proposed system architecture
NDN-D-PR	Proposed system architecture with Prefetch

Table I shows the different types of experiments that we used for the performance comparison. NDN-R represents the default NDN architecture with the default settings. NDN-D represents our proposed system architecture for cache management. NDN-D-PR represents the cache management with the prefetch feature enabled as explained in Section III-C.

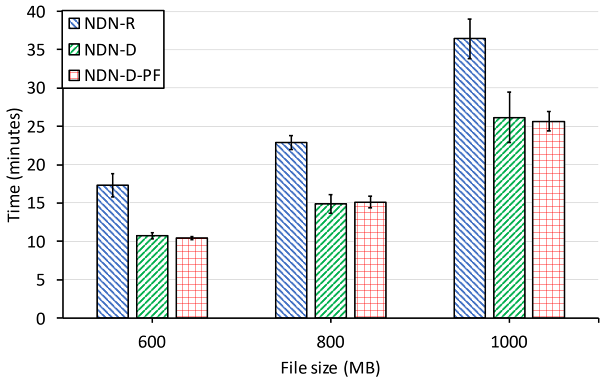
Figure 4a shows the transfer performance when the file is requested for the first time; in this scenario, the file is not cached in any router CS. Since in this work we are interested in caching the large file transfers, we run the experiments listed in Table I with three different file sizes; 600MB, 800MB, and 1GB. These file sizes are larger than the default router CS size of 500MB used in NDN. Figure 4a shows that the performance of NDN-R and NDN-D are similar since the file is not present in any routers' CS and the files are always requested from the NDN producer.

On the other hand, the NDN-D-PF shows a 13.5% – 23.6% performance improvement over other approaches. This performance gain is due to the file retrieval process in NDN-D-PF where the first part is requested normally through the path (Consumer -> R1 -> R2 -> producer) and simultaneously, the controller will direct routers R3 and R4 to prefetch the other parts of the file. Therefore, routers R3 and R4 satisfy the interests request for other file segments.

Figure 4b shows the transfer performance when the file is requested for the second time; in this case the file is already cached in the network (i.e., some router's CS). The NDN-R shows no benefit from the NDN router caching mechanism. This is due to the fact that file sizes are larger than the CS. Therefore, the cached file segments will be replaced with new file segments (similar to the example in Section III. In the NDN-R case, all file segments will be requested from the NDN producer. On the other hand, NDN-D and NDN-PF show 28.1% – 38% performance gains. This is due to the NDN-D and NDN-PF approaches where the files are cached on



(a) File Not cached.



(b) File cached.

Fig. 4: WAN Performance Evaluation of the distributed cache with/without prefetch. (a) when the file is not cached, and (b) when the file is cached.

multiple routers' CS and all file segments are served by the routers rather than the NDN producer.

Although our architecture focused on large dataset transfers, small file transfers will still follow the default NDN route (single path only). Small files are not split among several routers unless they are larger than the routers' CS. Our approach adds a small additional delay due to the communication with the controller. However, the delay is negligible as the controllers are typically one-hop away from the routers.

VI. Conclusions

Several scienfic and research domains deal with high volume data transfers including high-energy physics workflows such as CMS. Managing these large file transfers require a higher level of network management to cache and retrieve the files using NDN. In this paper, we presented an architecture that uses centralized control with NDN to provide faster in-network access to large datasets. In this work, we use SDN to provide an intelligent and efficient solution for data distribution and retrieval across multiple NDN routers' cache. The SDN controller is responsible for distributing and splitting large files in the network to fit multiple NDN routers' content stores. Our proposed system architecture results in a performance gain of 28.1% - 38%, in comparison to the current NDN architecture. Further, we developed a prefetch mechanism which improves the file transfer time if the file already cached in the network.

In future work, we will focus on the cache placement problem. In the present scenario, multiple routers have to be selected to cache large files. Choosing the best locality of the routers will improve file transfer performance. We will also study the effect of link bandwidths and their use to avoid congested links in the NDN network.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Number OAC-1541442.

References

[1] S. Chatrchyan, G. Hmayakyan, V. Khachatryanand *et al.*, "The CMS experiment at the CERN LHC," *JINST*, vol. 3, p. S08004, 2008.

[2] C. Grandi, B. Bockelman, D. Bonacorsi *et al.*, "CMS Distributed Computing Integration in the LHC sustained operations era," in *Journal of Physics: Conference Series*, vol. 331, no. 6. IOP Publishing, 2011, p. 062032.

[3] I. Bird, "Computing for the Large Hadron Collider," *Annual Review of Nuclear and Particle Science*, vol. 61, pp. 99–118, 2011.

[4] L. Zhang, A. Afanasyev, J. Burke *et al.*, "Named data networking," *ACM SIGCOMM CCR*, vol. 44, no. 3, pp. 66–73, 2014.

[5] B. A. A. Nunes, M. Mendonca *et al.*, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Comm. Surveys Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[6] H. Salah and T. Strufe, "Comon: An architecture for coordinated caching and cache-aware routing in CCN," in *Consumer Communications and Networking Conference (CCNC)*. IEEE, 2015, pp. 663–670.

[7] H. K. Rath, B. Panigrahi, and A. Simha, "On Cooperative On-Path and Off-Path Caching Policy for Information Centric Networks (ICN)," in *Advanced Information Networking and Applications (AINA), 2016 IEEE 30th International Conference on*. IEEE, 2016, pp. 842–849.

[8] Y. Xin, Y. Li, W. Wang *et al.*, "Content aware multi-path forwarding strategy in Information Centric Networking," in *Computers and Communication (ISCC), 2016 IEEE Symposium on*. IEEE, 2016, pp. 816–823.

[9] R. Chiocchetti, D. Perino, G. Carofiglio *et al.*, "Inform: a dynamic interest forwarding mechanism for information centric networking," in *Proc. 3rd ACM SIGCOMM workshop on ICN*. ACM, 2013, pp. 9–14.

[10] M. Alhowaidi, B. Ramamurthy *et al.*, "The Case for Using Content-Centric Networking for Distributing High-Energy Physics Software," in *ICDCS*, June 2017, pp. 2571–2572.

[11] H. Lim, A. Ni, D. Kim *et al.*, "NDN Construction for Big Science: Lessons Learned from Establishing a Testbed," *IEEE Network*, vol. 32, no. 6, pp. 124–136, Nov. 2018.

[12] H. Newman, A. Mughal, D. Kcira *et al.*, "High Speed Scientific Data Transfers Using Software Defined Networking," in *Proceedings of the Second Workshop on Innovating the Network for Data-Intensive Science*, ser. INDIS '15. New York, NY, USA: ACM, 2015, pp. 2:1–2:9, event-place: Austin, Texas.

[13] M. Alhowaidi, D. Nadig, B. Ramamurthy *et al.*, "Multipath Forwarding Strategies and SDN Control for Named Data Networking," in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS) (IEEE ANTS 2018)*, Indore, India, Dec. 2018.

[14] V. Lehman, A. Gawande, B. Zhang *et al.*, "An experimental investigation of hyperbolic routing with a smart forwarding plane in NDN," in *Intl. Sym. on Quality of Service (IWQoS)*, June 2016, pp. 1–10.

[15] J. Blomer, P. Buncic, and R. Meusel, "The CernVM file system," Technical Report, Tech. Rep., 2013.

[16] M. Berman, J. S. Chase, L. Landweber *et al.*, "GENI: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5 – 23, 2014, sI on Future Internet Testbeds - Part I.