SMURF: Efficient and Scalable Metadata Access for Distributed Applications from Edge to the Cloud

Bing Zhang
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Champaign, IL 61801 USA
bing@ncsa.illinois.edu

Tevfik Kosar
Department of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260 USA
tkosar@buffalo.edu

***Abstract*—In parallel with big data processing and analysis dominating the usage of distributed and cloud infrastructures, the demand for distributed metadata access and transfer has increased. In many application domains, the volume of data generated exceeds petabytes, while the corresponding metadata amounts to terabytes or even more. In this paper, we propose a novel solution for efficient and scalable metadata access for distributed applications across wide-area networks, dubbed SMURF. Our solution combines novel pipelining and concurrent transfer mechanisms with reliability, provides distributed continuum caching and prefetching strategies to sidestep fetching latency, and achieves scalable and high-performance metadata fetch/prefetch services in the cloud. We also study the phenomenon of semantic locality in real trace logs which is not well utilized in metadata access prediction. We implement our predictor based on this observation and compare it with three existing state-of-the-art prefetch schemes on Yahoo! Hadoop audit traces. By effectively caching and prefetching metadata based on the access patterns, our continuum caching and prefetching mechanism greatly improves local cache hit rate and reduces the average fetching latency. We replayed approximately 20 Million metadata access operations from real audit traces, in which our system achieved 80% accuracy during prefetch prediction and reduced the average fetch latency 50% compared to the state-of-the-art mechanisms.**

***Keywords*-Metadata access, semantic locality, prefetching, continuum caching, scalability, efficiency.**

I. Introduction

We are witnessing a new era that offers new opportunities to conduct data-intensive scientific research with the help of recent advancements in computational, storage, and network technologies. With the rapid deployment of distributed infrastructures and the collaborations between different organizations, it is feasible and promising to run scientific applications on these large-scale geo-distributed infrastructures. More recently, with numerous growth of Internet of Things (IoT) [1] devices (e.g., sensors, smartphones, smart vehicles, and smart homes, etc) connecting to the world, a paradigm shift from cloud to edge computing sparks the new needs and requirements in the field of cloud computing and distributed applications. In many application domains including environmental and coastal hazard prediction, climate modeling, high-energy physics, astronomy, and genome mapping, the volume of data generated already exceeds petabytes, while the corresponding metadata amounts to terabytes or even more [2]. According to

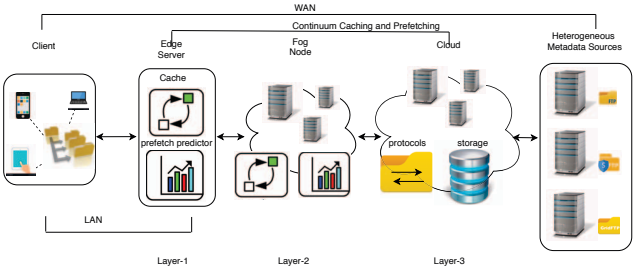

Fig. 1: Client devices fetch/prefetch metadata of interest in WAN via SMURF's distributed continuum cache and semantic prefetch mechanism.

Roselli [3]'s study, more than 50% of all I/O operations are due to metadata-intensive computing and the requests to read file attributes dominate in all workloads. Metadata transferring is extremely latency sensitive due to user experience and critical business operations: Google reported 20% revenue loss due to a specific experiment that increased the time to display search results by as little as 500 milliseconds; and Amazon reported a 1% sales decrease for an additional delay of as little as 100 milliseconds [4].

Unfortunately, most of the existing studies have focused on efficient and scalable transfer of large-scale data, and there has been little work focusing on the optimization of remote access and transferring of metadata [5] in wide-area networks. Considering wide-area network latency, the frequency of revalidation of metadata, and the rapid growth of IoT, an efficient and scalable metadata access and transfer mechanism is demanded and becomes a cornerstone of modern distributed IT infrastructures.

Semantic locality is a high-level abstraction of data access sequence locality, which is an extension to data spatial and temporal localities. Semantic locality potentially exists and can be captured when data access sequence is totally random, the frequency of duplicated data access is once or rare, or when a data access sequence is interleaved with large enough amount of "noisy neighbors". Especially, applications follow the convention to manage files under semantic folders and paths. Zhu et al [6] studies one of the most popular web-based version control hosting service Github and shows that the

978-1-7281-2708-8/19/$31.00 ©2019 IEEE
DOI 10.1109/EDGE.2019.00031

common semantic folders: lib, src, $test$, doc, and $examples$ are among top twenty most commonly used folders across 140,000 Github projects which suggests that different projects tend to follow a convention to create their folders and organize their files.

In this paper, we present a novel metadata access and retrieval system built on the distributed continuum caching and prefetching architecture to effectively fetch, prefetch, and cache metadata on different hierarchical layers (as shown in Figure 1) between clients and remote IO servers in WAN. The main contributions of this paper include:

- Design and implementation of an efficient and scalable metadata access and transferring technique for millions of metadata instances and records over WAN.
- Design and implementation of a distributed continuum caching and prefetching technique to sidestep metadata access and transfer latency in WAN.
- A solution that can work for heterogeneous metadata sources not just specific for any protocol.
- A study of the phenomenon of semantic locality in real system traces and development of a novel semantic locality prefetch scheme.
- Comparison of four different prefetch predictors (our semantic locality prefetch predictor and three state-of-the-art predictors, namely, NEXUS, AMP and FARMER) and the legacy LRU cache on the Yahoo HDFS traces.

The rest of the paper is organized as follows: Section II outlines existing relevant algorithms to provide a background for existing metadata prefetch schemes; Section III describes our proposed system architecture and discusses its design issues; Section IV presents the simulation methodology and performance evaluations; and Section V concludes the paper.

II. Background and Related Work

Metadata prefetching prediction [7]–[9] studies the history access sequence to predict the request patterns with different methodologies. NEXUS [8] applies a weighted-group-based prefetching algorithm for prediction. A weighted directed graph will be built on the fly when the metadata server (MDS) receives requests from clients. For a given request, NEXUS looks up the graph to predict top-k vertices with the largest edge weight as the best prefetching candidates. Experiments show that their prefetching prediction can effectively reduce clients' average response time with a reasonable overhead.

FARMER [10] further investigates how a request's attribute information affects the file successor probability. Authors statistically analyze the average probabilities for different trace sequences. Then they apply a linear combination model to consider the joint effect of the history access sequence and the semantic attributes of requests. FARMER builds a relationship graph between predecessor and successors in a certain size history window which is similar to NEXUS. It applies Integrated Path Algorithm (IPA) to detect the semantic attributes correlation between predecessor and each successor. The best prefetching accuracy that FARMER achieves is 64%, where NEXUS can achieve only 43%.

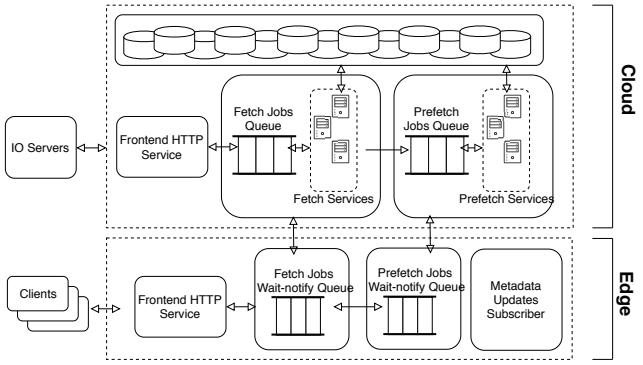

Fig. 2: High-level overview of SMURF's metadata fetching and prefetching services between edge server and cloud.

AMP [9] uses a different approach to predict the request patterns based on the studies of history access sequence. Authors apply N-gram [11] model which has been widely used in natural language processing to train the prediction model in a quasi-online fashion. Authors state that 3-gram with up to 6 prefetching items can give the best hit ratio with less computational overhead. The experiments show that AMP can outperform NEXUS by 4% on hit ratio and reduce the average response time by 8%.

III. System Architecture

In this section, we introduce the SMURF architecture, discuss each functional component of the system, and outline the details of the workflows in the system.

A. SMURF Overview

SMURF consists of a hierarchical architecture (as shown in Figure 2) with two major components: (1) a centralized cloud cluster with the scalable fetch/prefetch services provides the universal pipelining and concurrent metadata transfer mechanisms with reliability; (2) the distributed continuum caching and smart prefetching strategies have been deployed on edge/fog nodes in clients' nearby network.

B. Fetch/Prefetch Services

Cloud transfers the large-scale metadata using *concurrency* and *pipelining* by the instances of fetch/prefetch services running across machines in WAN. The details of sub-components and features are described below.

1) Metadata Transferring via a Cluster of Fetch/Prefetch Services: A fetch/prefetch service keeps at most one *singleton connection* to the remote server and only serves one request. With N services running in the cluster, the cloud establishes N TCP connections to remote servers and transfers N metadata requests concurrently. Cloud controls the *concurrency* level by managing the number of active services. When cloud deploys services across M machines, the cloud service can transfer N metadata requests from M machines which exceed the limitation and bottleneck of a single machine and tolerate $(M-1)$ machine failures.

2) Universal Transferring Stream: We designed and implemented the *universal* transferring stream to retrieve metadata of interest from heterogenous IO servers using *pipelining*, where the pipeline capacity defines the maximum number of C messages to be continuously sent over one TCP connection without waiting for the acknowledgement of previous requests. One request is organized as a chain of ordered pairs $\{cmd, parser\}$, where each protocol cmd is associated with a predefined $parser$. One request transferr is successful as long as all protocol commands have been sent and parsed correctly. Otherwise, this request will be regarded as a failure, and it will be either re-transferred or skipped according to the result of parser. Our protocol parser library supports FTP [12], SFTP [13], GRIDFTP [14], and IRODS [15].

3) Edge Server's Request and Response Multiplexing: We design and implement a *wait-and-notify queue* to efficiently send and receive messages between edge server and cloud. Multiple worker threads of an edge/fog server can enqueue the requests and wait for the notification of completion concurrently. Moreover, the deduplication of sending similar requests is executed on the edge/fog server to reduce the usage of WAN bandwidth and alleviate the overload of cloud. The queue can be configured to be "nowait" mode, thus threads will not wait for the completion of requests. This mode has been widely used in the prefetching work. The mechanism of wait-and-notify queue exhibits the high performance in multiple threading environment: message sending and receiving are designed to be interleaved between multiple threads.

4) Publish/Subscribe Metadata Updates: If the metadata has been updated in local, cloud can automatically publish the updated metadata to all edge servers which have fetched this metadata before.

C. Distributed Continuum Caching and Prefetching

The system consists of the distributed continuum cache layers-{1,2,3} in Figure 1, where the less amount of metadata with higher popularity will be cached closer to the clients. The prefetch predictors can be installed on the edge/fog nodes with the judicious parameters to retrieve the locality metadata into each layer's local cache. One edge server can fetch/prefetch metadata from the optional fog node's local cache, which can be denoted as F_{edge}/$\{P_{edge}\}$. The fog node sends the cached metadata back to the edge server or forward the cache miss fetch request F_{edge} and prefetch requests $\{P_{edge}\}$ to the cloud. The cache miss fetch request F_{edge} can cause the fog node's prefetch framework (more details in III-D) to consult its prefetch predictor on the aggressive prefetch $\{P_{fog}\}$. The overlapping prefetch requests between $\{P_{edge}\}$ and $\{P_{fog}\}$ requests will be de-duplicated by the wait-notify queue (discussed in III-B3) to reduce WAN bandwidth usages and alleviate the cloud overload and the fog node will send back the edge server's requested prefetch metadata $\{P_{edge}\}$ upon the completion.

D. Prefetch Framework

Each request will be sent to the prefetch predictor to analyze and build a prefetch correlation relationship. When a request causes a local cache miss, the prefetch framework will consult the prefetch predictor for the potential prefetching files. The prefetch framework checks whether each prefetch candidate exists in the current local cache,. If there is a cache miss on this candidate, the prefetch framework will pack a prefetch request with the information of this file (e.g., file path and priority) and then send it to the cloud. One prefetch request with a higher priority will preempt the lower priority request on the available prefetch services. The framework maintains the counters in LRU cache to denote how many cache misses happening on one file path, since the essence of Least Recent Used cache replacement algorithm is based on temporal locality and the cache miss information of the coldest file path need be replaced and cached out to reflect the temporal access locality and also reduce the memory usage.

E. Semantic Locality Prefetch Predictor

We designed and implemented a novel prefetch predictor based on the directory semantic locality. First of all, we will describe the workflow of semantic locality predictor and then we discuss the reason that semantic locality predictor is applicable for some types of real workloads. When a request on a file path f causes a local cache miss, predictor will check whether its parent file path f_p object is cached or not. (i) If the parent file path object has not been cached, then predictor will create an object of this parent file path and put it into the local cache by setting the value of the counter to 1. If parent file path object has been cached, then semantic locality predictor will read cache miss counter from the parent file path object and increase it by 1. If this cache miss counter exceeds threshold T, predictor will decide to prefetch metadata of parent file path f_p and sibling file paths $\{f_{si}\}$ under the parent file path. (ii) Otherwise, predictor will iterate the list of metadata of each subfile path f_{si} in the cache and send the prefetch requests of all cache missed subfile paths.On the semantic directory tree structure[1], semantic locality predictor can effectively match the pattern of workload, where most of subfiles under a common hotspot parent file path will be randomly accessed once or very few times. When data access sequence does not show the behavior of strong locality under a common file path, semantic locality predictor can set higher threshold to effectively prevent redundant miss-prefetching.

IV. Evaluation

A. Experiment Testbed and Traces

We conduct our experiments over Yahoo! Hadoop grid trace logs from Yahoo! Webscope dataset [16]. This trace consists of continuous daily metadata operations of Hadoop name node throughout the year of 2010. Geographical locations of the servers used in our experiments and the network specifications between them are presented in Figure 3.

[1]We define **semantic directory tree** as a storage system that organizes files into a logical file namespace.

Fig. 3: Experiment's network map and specifications.

1) Partial Trace File System Directory Tree Reconstruction: In trace logs, file path f always associates with types of operations, e.g., $open$, ls, $delete$, etc. Metadata read operations are directory tree idempotent[2] (e.g., $open$ and ls) and will not change the directory tree structure on trace file system. The write operations (e.g., $rename$, $delete$) can change trace file system directory tree dynamically. Yahoo! Webscope dataset encrypts each segment of the file path into 28 bytes string. Thus the approximate directory tree size of each Hadoop trace log on disk will be more than 200GB. Considering multiple log files in the trace, a very large disk capacity is needed to extract and analyze all at once. Hence, we decided to reconstruct trace file system directory tree partially. In this partial directory tree $T_{partial}$, we extract file path f_{list} from each $listStatus$ command and collect its sibling subfiles $\{f_{sub}\}$ from $open$, ls, $rename$ and $mkdirs$ commands, which can be expressed in $T_{partial} = \{f_{list}\} \cup \{f_{sub}\}$. This is the approximate emulation of trace file system directory tree structure in our prediction experiments, since the prefetch candidates of Farmer, Nexus and AMP are from the set of $listStatus$ file paths. And our directory locality semantic predictor can potentially prefetch the sibling files of $listStatus$ operation that have been constructed in the partial directory tree structure.

B. Scalability of Fetch/Prefetch Services

To emulate concurrent metadata transferring performance from remote servers, we turn off the caching and prefetching effects in the testbed, and then we continuously send a large number of requests from the client to the cloud and evaluate the latency distribution with the different number of fetching services. In Figure 4, with more number of services in the cloud, most of the requests can be done concurrently and the latency of each request is almost the same. Figure 5 demonstrates the scalability of prefetching services. We sent 10,000 and 100,000 prefetch requests and calculated the total prefetching elapse time on the edge server. With 85 prefetch services, the total latency of 100,000 prefetching requests can be completed within 60 seconds (0.6 millisecond per request on average) nearly 11 times faster than the prefetching with five prefetch services.

[2]We define **directory tree idempotent operation** as a metadata operation that can be executed once or repeatedly without mutating the parent and subfiles tree structure .

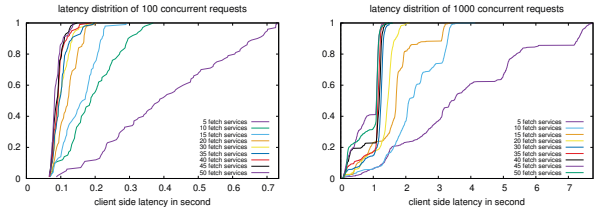

Fig. 4: Client fetch latency distribution for SMURF.

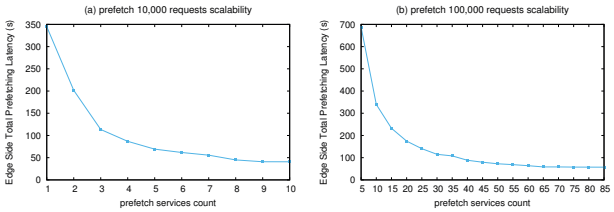

Fig. 5: Scalability of SMURF prefetch services.

C. Performance of the Hierarchical Layers

In hierarchical caching, the client sends fetching requests to the nearby edge server and the edge server acts as a middle caching and prefetching layer between client and cloud. We use the abbreviation term "EC" to stand for this and the "EFC" for the extra fog node between an edge server and cloud.

1) Cache Hit Rate: Figure 6(a) compares the cache hit rate between LRU cache and different prefetching schemes. In Figure 6(a), it is apparent that our directory locality scheme (denoted as "DLS") outperforms all other schemes on cache hit rate. It can achieve around 80%+ on all individual Hadoop audit logs because directory locality scheme can successfully capture the access pattern of "listStatus" operation in Yahoo! Webscope Hadoop audit log. AMP is another prediction scheme with high prediction rate which can achieve around 65%+ prediction rate. We also found that the cache hit rates of Nexus and Farmer are almost the same as that of LRU cache, since the prefetching candidates suggested by the prediction schemes of Nexus and Farmer are all from the history requests, while the Hadoop audit log exhibits a very skew popularity access in "listStatus" metadata operation. Most of "listStatus" operations just execute on a file path once or in rare times, which inevitably cause the low prediction rate of prediction schemes which are based on history access sequence. We also evaluated the cache hit rate performance with the different cache sizes. The experimental results show that there is no obvious performance enhancement with the increasing cache size (10%, 20% and 30%) when the prefetch scheme prediction rate is low.

2) Average Fetch Latency: We conducted experiments to measure the accumulated overhead of metadata transferring in the hierarchical layers without any cache and prefetch installed and represent them in three horizontal lines in Figure 6(b). The prefetching scheme with higher prediction rate can greatly

reduce the average fetch latency, since most of metadata can be accessed locally. Nexus and Farmer's average latencies are above the the bar. We think that is due to two factos: 1) RTT between client and remote IO server is around 32 milliseconds, while the accumulated RTT of EFC is above 40 milliseconds; 2) Nexus and Farmer predication rates are the same as that of LRU cache but the overhead of constructing and updating the relation graph in Nexus and Farmer prefetch schemes is not ignorable. In the setting of EFC, the deployment of a fog node is necessary in the scenario that the capacity of the computation and cache size on the edge server is limited. By simulating this scenario, we limit the cache size of edge node to 10%, and set the cache size of the fog node to 30% of the total number of "listStatus" metadata operations. In EFC hierarchical layers, DLS still outperforms all other schemes.

V. Conclusion and Future Work

We have presented an efficient metadata prefetching and caching system, called SMURF, and conducted experiments with a large amount of metadata requests using the real system traces (Yahoo! Hadoop) in WAN. Our experimental results show that SMURF can achieve 80% accurate prediction rate on "listStatus" metadata operation and reduce the average fetch latency up to 50% compared to other state-of-the-art mechanisms. SMURF's continuum caching and prefetching mechanism retrieves necessary enough popularity metadata closer to the location of clients, and it is friendly to IoT networks, where IoT devices with limited computing and storage capabilities can always access metadata locally from the proximity edge/fog compute nodes.

Acknowledgements

This project is in part sponsored by the National Science Foundation (NSF) under award number OAC-1724898.

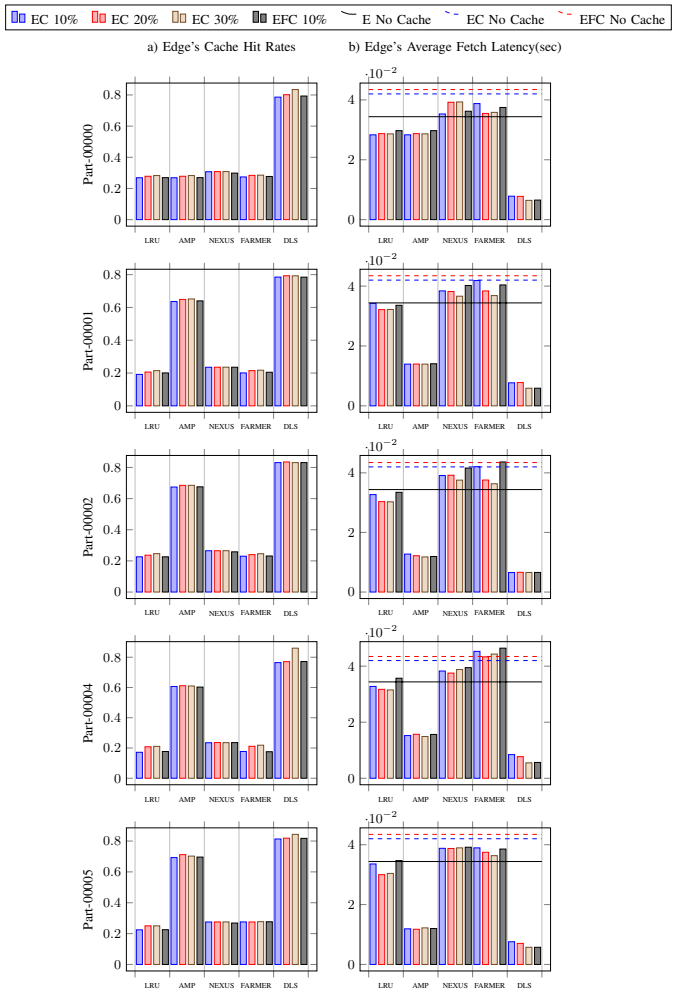

Fig. 6: Cache hit rate comparison of SMURF DLS with different prediction schemes on Yahoo! Hadoop trace.

References

[1] K. Ashton *et al.*, "That internet of things thing," *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[2] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller, "Dynamic metadata management for petabyte-scale file systems," in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2004, p. 4.

[3] D. S. Roselli, J. R. Lorch, T. E. Anderson *et al.*, "A comparison of file system workloads." in *USENIX Annual Technical Conference, General Track*, 2000, pp. 41–54.

[4] R. Kohavi, R. M. Henne, and D. Sommerfield, "Practical guide to controlled experiments on the web: Listen to your customers not to the hippo," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '07. New York, NY, USA: ACM, 2007, pp. 959–967. [Online]. Available: http://doi.acm.org/10.1145/1281192.1281295

[5] B. Zhang, B. Ross, and T. Kosar, "Dls: a cloud-hosted data caching and prefetching service for distributed metadata access," *International Journal of Big Data Intelligence*, vol. 2, no. 3, pp. 183–200, 2015.

[6] J. Zhu, M. Zhou, and A. Mockus, "Patterns of folder use and project popularity: A case study of github repositories," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. New York, NY, USA: ACM, 2014, pp. 30:1–30:4.

[7] J. Li and S. Wu, "Real-time data prefetching algorithm based on sequential patternmining in cloud environment," in *Industrial Control and Electronics Engineering (ICICEE), 2012 International Conference on*. IEEE, 2012, pp. 1044–1048.

[8] P. Gu, Y. Zhu, H. Jiang, and J. Wang, "Nexus: a novel weighted-graph-based prefetching algorithm for metadata servers in petabyte-scale storage systems," in *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, vol. 1. IEEE, 2006, pp. 8–pp.

[9] L. Lin, X. Li, H. Jiang, Y. Zhu, and L. Tian, "Amp: An affinity-based metadata prefetching scheme in large-scale distributed storage systems," in *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, 2008, pp. 459–466.

[10] P. Xia, D. Feng, H. Jiang, L. Tian, and F. Wang, "Farmer: a novel approach to file access correlation mining and evaluation reference model for optimizing peta-scale file system performance," in *Proceedings of the 17th international symposium on High performance distributed computing*. ACM, 2008, pp. 185–196.

[11] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[12] J. Postel and J. Reynolds, "Rfc 959: File transfer protocol (ftp)," *InterNet Network Working Group*, 1985.

[13] J. A. Moore, J. M. Johnson, S. F. T. P. Initiative *et al.*, "Transportation, land use and sustainability," 1994.

[14] W. Allcock, *Gridftp protocol specification*, Global grid forum, 2003, gFD.20.

[15] "The integrated rule oriented data system iRODS," http://www.irods.org/.

[16] http://research.yahoo.com.