IEEE 2nd International Conference on Cloud Computing Technology and Science

Correlation based File Prefetching Approach for Hadoop

Bo Dong[1], Xiao Zhong[2], Qinghua Zheng[1], Lirong Jian[2], Jian Liu[1], Jie Qiu[2], Ying Li[2]

[1]Xi'an Jiaotong University

[2]IBM Research-China

Agenda

1. Introduction
2. Challenges to HDFS Prefetching
3. Correlation based file prefetching for HDFS
4. Experimental evaluation
5. Conclusion

Motivation (1/2)

- Hadoop is prevalent!
  - *Internet service file systems* are extensively developed for data management in large-scale Internet services and cloud computing platforms.
  - *Hadoop Distributed File System* (HDFS), which is a representative of Internet service file systems, has been widely adopted to support diverse Internet applications.
- Latency of data accessing from file systems constitutes the major overhead of processing user requests!
  - Due to the access mechanism of HDFS, access latency of reading files from HDFS is serious, especially when accessing massive small files.

Motivation (2/2)

- Prefetching
  - Prefetching is considered as an effective way to alleviate access latency.
  - It hides visible I/O cost and reduces response time by exploiting access locality and fetching data into cache before they are requested.
- HDFS currently does not provide prefetching function.
  - It is noteworthy that users usually browse through files following certain access locality when visiting Internet applications.
  - Consequently, correlation based file prefetching can be exploited to reduce access latency for Hadoop-based Internet applications.
  - Due to some characteristics of HDFS, HDFS prefetching is more sophisticated than traditional file system prefetchings.

Contributions

- A two-level correlation based file prefetching approach is presented to reduce access latency of HDFS for *Hadoop-based Internet applications*.
- Major contributions
  - A two-level correlation based file prefetching mechanism, including metadata prefetching and block prefetching, is designed for HDFS.
  - Four placement patterns about where to store prefetched metadata and blocks are studied, with policies to achieve trade-off between performance and efficiency of HDFS prefetching.
  - A dynamic replica selection algorithm for HDFS prefetching is investigated to improve the efficiency of HDFS prefetching and to keep the workload balance of HDFS cluster.

Agenda

1. Introduction
2. Challenges to HDFS Prefetching
3. Correlation based file prefetching for HDFS
4. Experimental evaluation
5. Conclusion

Challenges to HDFS Prefetching (1/2)

- Because of the master-slave architecture, HDFS prefetching is two-leveled.
  - To prefetch metadata from NameNode and to prefetch blocks from DataNodes.
  - The consistency between these two kinds of operations should be guaranteed.
- Due to the direct client access mechanism and curse of singleton, NameNode is a crucial bottleneck to performance of HDFS.
  - The proposed prefetching scheme should be lightweight and help mitigate the overload of NameNode effectively.

Challenges to HDFS Prefetching (2/2)

- Invalidated cached data
  - When failures of blocks or DataNodes are detected, HDFS automatically creates alternate replicas on other DataNodes.
  - In addition, for sake of balance, HDFS dynamically forwards blocks to other specified DataNodes.
  - These operations would generate invalidated cached data, which need to be handled appropriately.
- The determination of what and how much data to prefetch is non-trivial because of the data replication mechanism of HDFS.
  - Besides closest distances to clients, replica factor and workload balance of HDFS cluster should also be considered.

Agenda

1. Introduction
2. Challenges to HDFS Prefetching
3. Correlation based file prefetching for HDFS
4. Experimental evaluation
5. Conclusion

Overview of HDFS prefetching (1/3)

- In HDFS prefetching, four fundamental issues need to be addressed
  - What to prefetch
  - How much data to prefetch
  - When to trigger prefetching
  - Where to store prefetched data
- What to prefetch
  - Metadata and blocks of correlated files will be prefetched from NameNode and DataNodes respectively.
  - *Metadata prefetching* reduces access latency on metadata server and even overcomes the per-file metadata server interaction problem
  - *Block prefetching* is used to reduce visible I/O cost and even mitigate the network transfer latency.

Overview of HDFS prefetching (2/3)

- What to prefetch
  - What to prefetch is predicated in file level rather than block level.
  - Based on file correlation mining and analysis, correlated files are predicted.
  - When a prefetching activity is triggered, metadata and blocks of correlated files will be prefetched.
- How much data to prefetch
  - The amount of to-be-prefetched data is decided from three aspects: file, block and replica.
  - In file level, the number of files is decided by specific file correlation prediction algorithms.
  - In block level, all blocks of a file will be prefetched.
  - In replica level, how many metadata of replicas and how many replicas to prefetch need to be determined.

Overview of HDFS prefetching (3/3)

- When to prefetch
  - When to prefetch is to control how early to trigger the next prefetching.
  - Here, the *Prefetch on Miss strategy* is adopted.
  - When a read request misses in metadata cache, a prefetching activity for both metadata prefetching and block prefetching will be activated.

Placement patterns about where to store prefetched data (1/2)

- Four combinations exist
  - Prefetched metadata can be stored on NameNode or HDFS clients.
  - Prefetched blocks can be stored on DataNodes or HDFS clients.

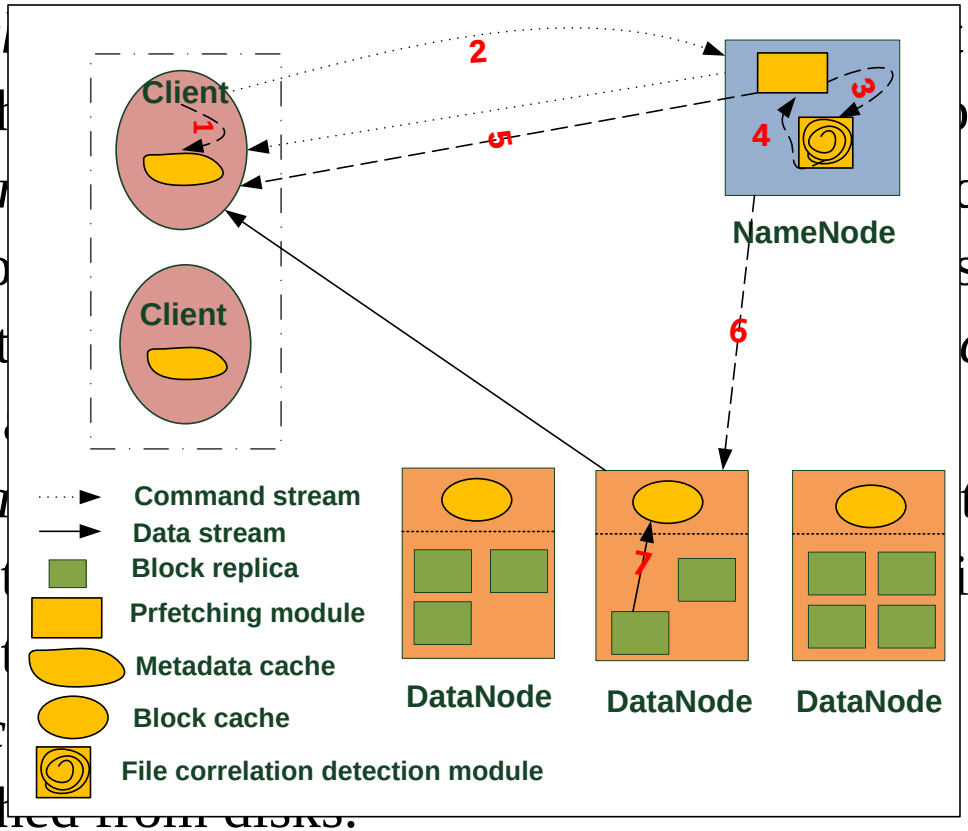
Features	NC-pattern	CC-pattern	ND-pattern	CD-pattern
Where to store prefetched metadata	NameNode	HDFS clients	NameNode	HDFS clients
Where to store prefetched blocks	HDFS clients	HDFS clients	DataNodes	DataNodes
Reduce the access latency on metadata server	Y	Y	Y	Y
Address the per-file metadata server interaction problem	N	Y	N	Y
Mitigate the overhead of NameNode	Maybe	Y	Maybe	Y
Reduce the visible I/O cost	Y	Y	Y	Y
Reduce the network transfer latency	Y	Y	N	N
Aggravate the network transfer overhead	Y	Y	N	N

Placement patterns about where to store prefetched data (2/2)

- A strategy to determine where to store prefetched metadata and blocks is investigated.
  - According to features of the above four patterns, when deciding which one is adopted, the following factors are taken into account: accuracy of file correlation prediction, resource constraint and real-time workload of HDFS cluster and clients.
- Four policies (the last one will be introduced later).
  - When the main memory on a HDFS client is limited, ND-pattern and CD-pattern are more appealing, since in the other two patterns, cached blocks would occupy a great amount of memory.
  - When accurate of correlation prediction results is high, NC-pattern and CC-pattern are better choices, because cached blocks on a client can be accessed directly in subsequent requests.
  - In the case of high network transfer workload, ND-pattern and CD-pattern are adopted to avoid additional traffic.

Architecture

- CD-pattern prefetching is taken as the representative to present the architecture and process of HDFS prefetching.
- HDFS prefetching is composed of four core modules.
  - *Prefetc* deals with prefetch of prefetching.
  - *File cor* d on NameNode. It is respo sed on file correlat *correlation mining engine*
  - *Metada* tores prefetched metadat including the metadat
  - *Block c* blocks which are prefetched from disks.



Dynamic replica selection algorithm (1/2)

- A dynamic replica selection algorithm is introduced for HDFS prefetching to prevent “hot” DataNodes which frequently serve prefetching requests.
- Currently, HDFS decides the replica sequences by physical distances (or rather rack-aware distances) between the requested client and DataNodes where all replicas of the block locate.

$$S = \arg \min_{d \in D} \{ D_d \} \qquad (1)$$

  - D_d: The distance between a DataNode *d* and a specified HDFS client
  - It is a static method because it only considers physical distances.

Dynamic replica selection algorithm (2/2)

- Dynamic replica selection algorithm
  - Besides distances, workload balance of HDFS cluster should also be taken into account.
  - Specifically, to determine the closest replicas, more factors are deliberated, including the capabilities, real-time workloads and throughputs of DataNodes.

$$S = \arg \min_{d \in D} \{ f(\frac{D_d, C_d}{W_d, T_d}) \} \qquad (2)$$

Dd	The distance between a DataNode d and a specified HDFS client
Cd	The capability of a DataNode d, including its CPU, memory and so on
Wd	The workload of a DataNode d, including its CPU utilization rate, memory utilization rate and so on
Td	The throughput of a DataNode d

Agenda

1. Introduction
2. Challenges to HDFS Prefetching
3. Correlation based file prefetching for HDFS
4. Experimental evaluation
5. Conclusion

Experimental environment and workload

- The test platform is built on a cluster of 9 PC servers.
  - One node, which is IBM X3650 server, acts as NameNode. It has 8 Intel Xeon CPU 2.00 GHz, 16 GB memory and 3 TB disk.
  - The other 8 nodes, which are IBM X3610 servers, act as DataNodes. Each of them has 8 Intel Xeon CPU 2.00 GHz, 8 GB memory and 3 TB disk.
- File set
  - A file set which contains 100,000 files is stored on HDFS. The file set, whose total size is 305.38 GB, is part of real files in BlueSky.
  - These files range from 32 KB to 16,529 KB, and files with size between 256 KB and 8,192 KB account for 78% of the total files.
- Accuracy rates of file correlation predictions
  - In the experiments, accuracy rates are set to 25%, 50%, 75% and 100% respectively.

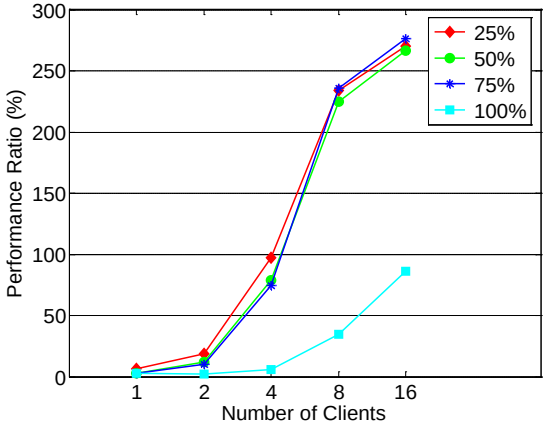
Experiment results of original HDFS

- Aggregate read throughputs
  - They do not always grow as numbers of HDFS clients in a server increase.
  - Conversely, the growth of throughputs is not obvious when client numbers increase from 4 to 8, and throughputs even decrease when numbers increase from 8 to 16.
  - When the number is 8, throughput grows up to the maximum.

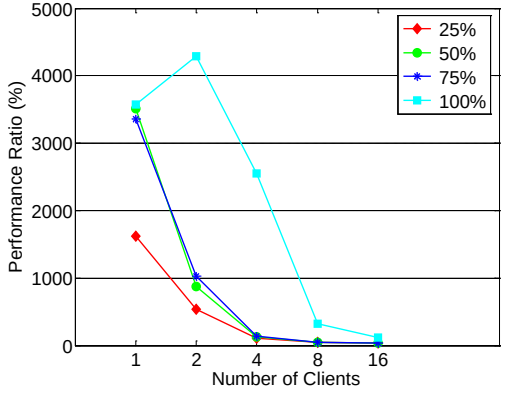
Number of HDFS clients in a server	Total Download time of 100 files (Sec)	Aggregate read throughput (MB/Sec)	CPU utilization rate of NameNode	Network traffic (MB/Sec)

The results of original HDFS experiments in different client numbers are treated as base values. Performances of HDFS prefetchings are compared with these base values which are tested in the same client number.

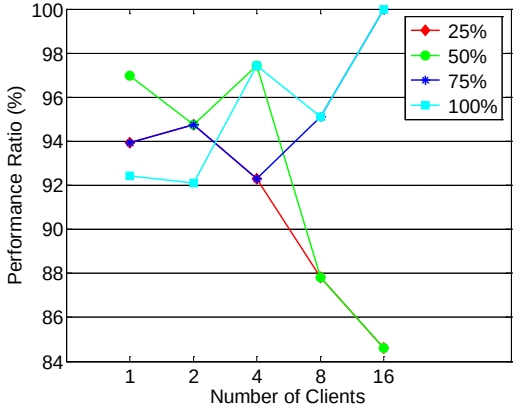
Experiment results of NC-pattern prefetching



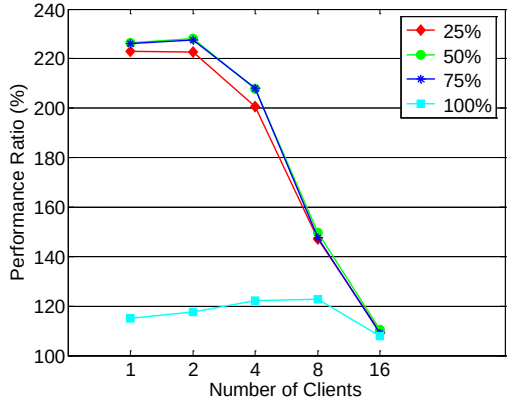
Total Download Time



Aggregate Read Throughput



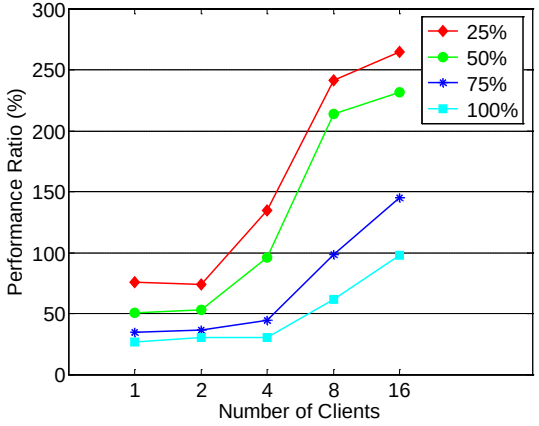
Average CPU Utilization of NameNode



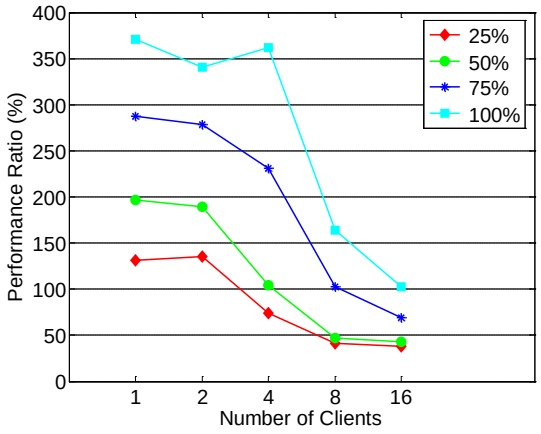
Network Traffic

- NC-pattern prefetching is aggressive.
- It prefetches blocks to clients so as to alleviate the network transfer latency.
- Numbers of HDFS clients and accuracy rates of file correlation predictions greatly influence the performances.
- When client numbers are large and accuracy rates are low, aggregate throughputs are reduced by 60.07%.
- On the other hand, when client numbers are small and accuracy rates are high, NC-pattern prefetching generates an average 1709.27% throughput improvement while using about 5.13% less CPU.

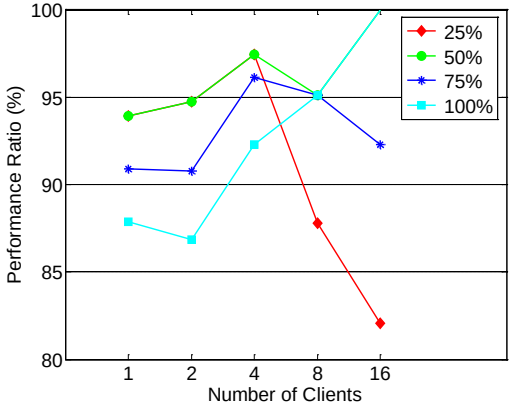
Experiment results of CC-pattern prefetching



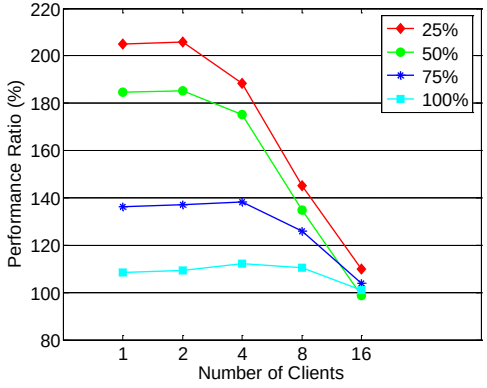
Total Download Time



Aggregate Read Throughput



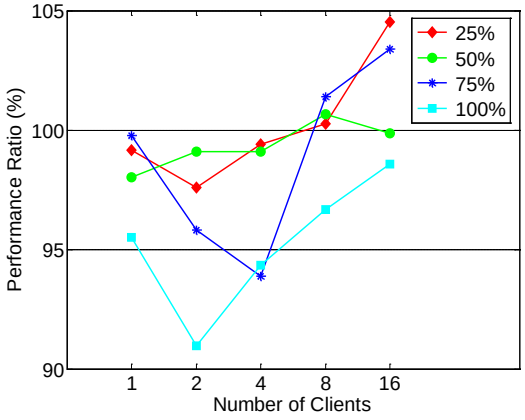
Average CPU Utilization of NameNode



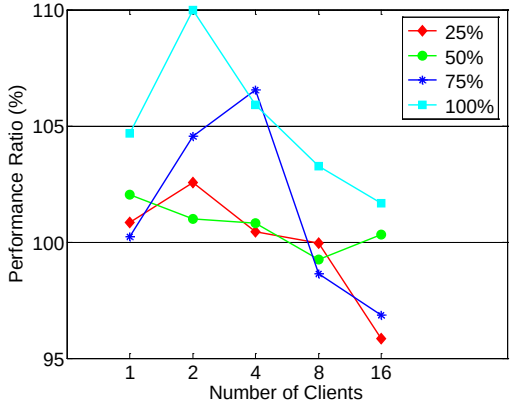
Network Traffic

- CC-pattern prefetching is aggressive.
- It prefetches blocks to clients so as to alleviate the network transfer latency.
- Numbers of HDFS clients and accuracy rates of file correlation predictions greatly influence the performances.
- When client numbers are large and accuracy rates are low, aggregate throughputs are reduced by 48.04%.
- On the other hand, when client numbers are small and accuracy rates are high, CC-pattern prefetching generates 114.06% throughput improvement while using about 6.43% less CPU.

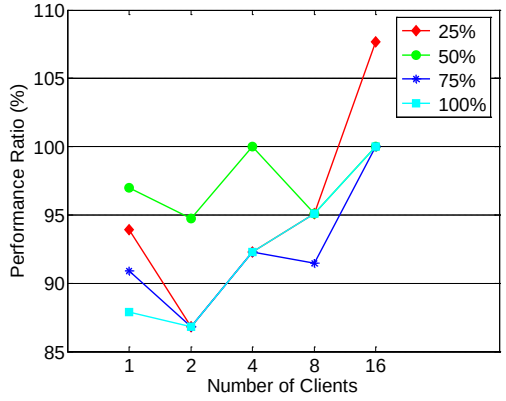
Experiment results of ND-pattern prefetching



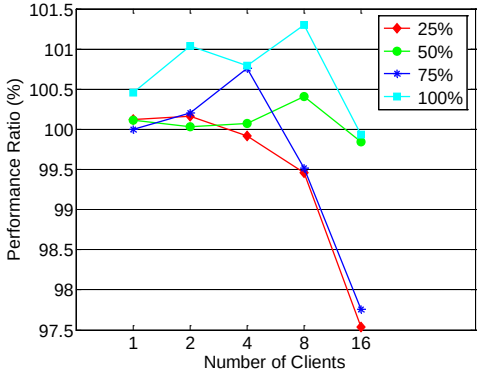
Total Download Time



Aggregate Read Throughput



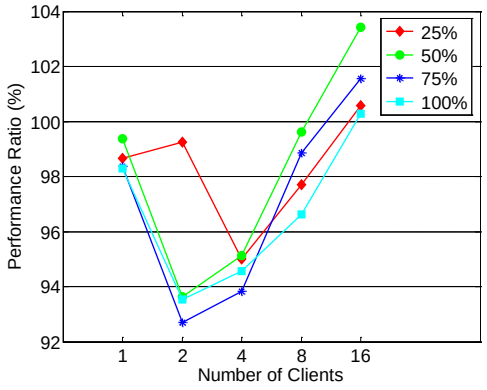
Average CPU Utilization of NameNode



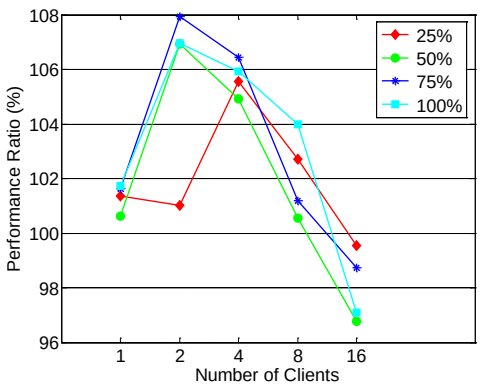
Network Traffic

- ND-pattern prefetching is conservative.
- Compared with NC-pattern and CC-pattern, it generate relatively small but stable performance enhancements.
- For situations aggregate read throughputs are improved, it generates an average 2.99% throughput improvement while using about 6.87% less CPU.
- Even in situations which aggregate read throughputs are reduced, average reductions of throughputs are just 1.90% in ND-pattern.
- Moreover, numbers of HDFS clients and accuracy rates of file correlation predictions have trivial influence on the performance.

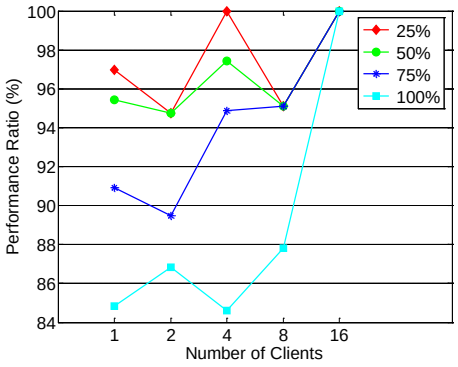
Experiment results of CD-pattern prefetching



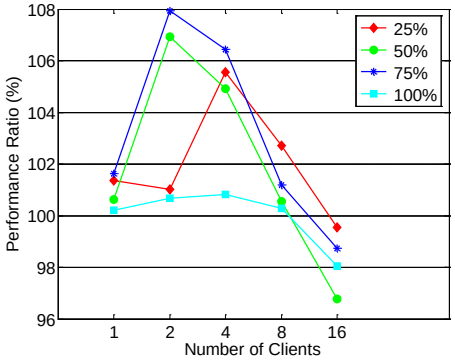
Total Download Time



Aggregate Read Throughput



Average CPU Utilization of NameNode



Network Traffic

- CD-pattern prefetching is conservative.
- Compared with NC-pattern and CC-pattern, it generate relatively small but stable performance enhancements.
- For situations aggregate read throughputs are improved, it generates 3.72% throughput improvement while using about 7.25% less CPU.
- Even in situations which aggregate read throughputs are reduced, average reductions of throughputs are just 1.96%.
- Moreover, numbers of HDFS clients and accuracy rates of file correlation predictions have trivial influence on the performance.

Placement patterns about where to store prefetched data

- Based on the above experiment results, the fourth policy on how to choose patterns of HDFS prefetching is obtained.
  - When correlation prediction results are unstable, conservative prefetching patterns (ND-pattern and CD-pattern) are more appropriate because they can avoid the performance deterioration due to low accuracy rates.

Agenda

1. Introduction
2. Challenges to HDFS Prefetching
3. Correlation based file prefetching for HDFS
4. Experimental evaluation
5. Conclusion

Conclusion

- Prefetching mechanism is studied for HDFS to alleviate access latency.
  - First, the challenges to HDFS prefetching are identified.
  - Then a two-level correlation based file prefetching approach is presented for HDFS.
  - Furthermore, four placement patterns to store prefetched metadata and blocks are presented, and four rules on how to choose one of those patterns are investigated to achieve trade-off between performance and efficiency of HDFS prefetching.
  - Particularly, a dynamic replica selection algorithm is investigated to improve the efficiency of HDFS prefetching.
- Experimental results prove that HDFS prefetching can significantly reduce the access latency of HDFS and reduce the overload of NameNode, therefore improve performance of Hadoop-based Internet applications.


Thank you.