Issues in Information Systems
Volume 25, Issue 2, pp. 309-324, 2024

DOI: https://doi.org/10.48009/2_iis_2024_125

A fog-based technique for optimized distributed cloud cache

Nader Mohamed, *Pennsylvania Western University, mohamed@pennwest.edu*
Jameela Al-Jaroodi, *Robert Morris University, aljaroodi@rmu.edu*

Abstract

Cloud data centers can face many challenges at certain times due to the huge number of requests to access and download specific data sets or files for software, games, movies, and several other types of data as they are released for thousand or millions of clients. Integrating both cloud computing and fog computing can solve this problem. The main question here is how to utilize both technologies to provide efficient data storage and fast data transfer for the clients. Here we propose the use of distributed cloud cache, to help solve this problem. This distributed cloud cache uses small cache nodes and dual direction parallel transfer techniques to reduce the overload on the cloud data centers. This involves creating multiple small cache nodes that will help a data center partition the requested data set or file across these nodes and the data center itself. These small cache nodes can be distributed over multiple locations on the Internet to balance the ongoing download rates and to reduce the overhead on the cloud data center. The distributed cache nodes can be mounted over some dedicated servers or on fog platforms. We implemented and evaluated the proposed distributed cloud cache, and the results show considerable performance gains.

Keywords: Cloud computing, fog computing, distributed cache, data transfer

Introduction

Achieving high performance in distributed computing environments relies on various factors including processing power; storage, memory, and network characteristics; and the algorithms and software architectures used. When large volumes of data are needed across multiple locations, data transfer rates and latencies become an important factor. In cloud computing, various applications rely on cloud storage and data centers to hold their data and allow multiple users to access this data when needed. One of the major challenges facing the cloud in this regard is the data transfer bottleneck (Armbrust et al., 2010). In addition, users also flock to certain software or media content providers in huge numbers to download software, games, movies, and several other types of data as they are released. Generally, this type of data is huge and takes a long time to download. This creates immense load on these providers and eventually leads to long delays and lower transfer rates. Cloud service providers and data/software distributers strive to satisfy their clients' quality of service needs and ensure the availability and reliability of their services. In addition, it is important to make sure that requested data sets are transferred within acceptable time frames. Data replication and geographical distribution across various locations as that used in SPANStore (Wu et al., 2013) can help in this regard; however, more can be achieved by incorporating new techniques and methods in data organization and transfer.

Since data is usually available in remote locations on the cloud, one possible approach is to bring the data closer to the client. Creating more replicas at locations near the clients will improve clients' access to this data. However, this may not be achievable at all locations, and it is expensive to have too many replicas

just to get closer to the clients. Furthermore, if demand increases within one location, the replicas in that location will be overloaded, while other servers have nothing to do. Partial replicas may also be used to reduce storage needs (Shen et al., 2015). However, many of these are specialized and require complex management techniques to make sure that the data segments needed are always available for the clients at the right locations. In addition, when the client requests partitions that are not currently available close by, more delays will be experienced as the data is found and retrieved from other locations. Some of these issues can be mitigated if better and faster data transfer techniques are used so that the distance becomes less of an issue. Combined with suitable mechanisms to partition, distribute and manage data sets, this may lead to better overall performance.

The main contribution of this paper is proposing an optimized distributed cloud cache using dual direction parallel transfer techniques to reduce overload on data centers and distribute it across multiple points of transfer. The approach relies on creating multiple cache nodes on multiple fog nodes that will help a data center partition the download across multiple cache nodes and the data center itself. The dual direction transfer will provide automatic load balancing across all download nodes, while a cache management model will handle the different aspects of caching the data and exchanging data sets based on current demand. The technique will help distribute the data download work evenly across all available cache nodes, while minimizing the client involvement in the process. Dual direction data transfer has been designed and evaluated and has demonstrated significant gains in terms of overall transfer times (Mohamed et al., 2013). However, the use of the technique with an appropriate caching mechanism will help improve overall data centers performance in high demand situations.

One issue to clarify though is that this technique is mostly suitable for large data sets that are written once and read many times. Infrequent changes and updates can be accommodated, yet it will not work well with data sets that are updated continuously and in real-time as we run into data integrity and accuracy issues across the available cached partitions. With this in mind, we still foresee huge advantages to a large set of data types that conform to this condition. For example, when a gaming company releases a new game that is several hundred GBs in size, thousands of users will be trying to download that game as soon as it is released. Latency in the download process could extend to hours for many of these users. For example, when released, Call of duty Black Ops 3 sold almost 10 million copies in three days (Makuch, 2015). Most of these sales were online and required downloading the game which takes over 50GB of space. Many complained of extended download times and very long waits during that period.

Similarly, data is collected by the scientific communities through experiments and observations generating TBs of data that is later accessed by hundreds of researchers for analysis (Abdalla, 2022). This type of data has similar characteristics and its transfer to the various users can benefit from the proposed method. WLCG (Worldwide LHC Computing Grid) offers experimental data generated by the Large Hadron Collider (LHC) at CERN (CERN, 2024) to over 170 computing centers for analysis. Another example is the Earth System Grid (ESG) (Williams et al., 2009), which provides huge volumes of climate data for analysis by researchers and weather forecasters.

Related Work

In this section we discuss some relevant work in the areas of cache technologies and management techniques. The cloud has become a major player in providing storage and data exchange services for various domains. Commercial and private implementations are available, where clients can access and download the required data sets. However, a lot of research conducted show the challenges and limitations of these services. For example, (Armbrust et al., 2010) describes ten challenges, two of which pertain directly to storage and use. These are the data transfer bottleneck and the storage scalability. Some

researchers studied the characteristics and access patterns of cloud data storage systems using an actual cloud-based file system (Liu et al., 2013). One of the observations is the low read/write ratio, and high percentage of read only content. Many tried to address these issues; however, many of the proposed solutions lead to other challenges. For example, the use of replication or partial replication such as the examples described in (Wu et al., 2013; Shen et al., 2015). In most cases, the replication is used to increase reliability and allow for faster access based on location or load levels. However, most suffer from other problems such as load balancing overhead, unavailable content at certain locations or high demand on some locations while others are not in use.

Some effort is also invested in assisting the process through caching. However, the majority of work we came across addresses memory caching to enhance data access and processing performance rather than the data transfer performance. Several examples are available such as the Cache-as-a-Service model (Han et al., 2011) that allocates dedicated remote servers with large memory as memory caches for cloud data and applications. The advantage is enhanced performance via increased available cache size and as a source of revenue for the cloud service providers. However, this approach does not improve data transfer performance. A similar approach is discussed in (Kantere et al., 2011), where a pricing policy is introduced for using cloud cache to enhance performance. In addition, as part of their study of data access behavior in (Liu et al., 2013) the authors discuss caching to improve access in cloud-based file systems.

In practice, several cloud service providers also use in-memory cache to enhance performance. Amazon ElastiCache (ElastiCache, 2024) uses in-memory caches to allow web applications to access data faster than going directly to the disc-based databases. It uses multiple nodes dedicated for in-memory caches that clients can use to create and use data stores and caches for their web applications. Amazon ElastiCache for Redis is designed to be compliant with the Redis memory service (Redis, 2024). Amazon also makes its cache service compliant with Memcached (Memcached,2024), which is a memory object caching system.

Several others discussed caching on the cloud and cache management technique. For example, client-side storage is proposed in (Arteaga, 2012) to enhance the performance of VM (Virtual Machine) data access using block-based data transfer. The model relies on allowing multiple VMs on the same hot to use the same cache while avoiding data corruption. Distributed cache for Hadoop file system (Zhang et al., 2012) offers another way to better manage caching and improve overall performance when handling large data sets.

Caching mechanisms and management techniques have improved significantly. However, most work addresses in memory caching targeted towards improving processing performance for large data sets. However, very little work (as far as we have found) addresses the data transfer performance using caching techniques. As a result, there is much to be achieved when we can apply similar concepts at the data transfer level. Our work explores this possibility and introduces a caching mechanism targeted to enhancing the download performance of large data sets. More specifically, for could data providers who provide large data sets to many clients.

As we investigate the different models, and techniques introduced, we realized that there are many areas where improvements are possible. One is the better utilization of proven techniques used in other contexts to improve transfer rates. We explore the use of caching mechanisms combined with high throughput downloads to enhance overall download times from the cloud. As a result, we can combine some of the benefits of these approaches to achieve better results for specific types of data sets and files that many clients need to retrieve from the cloud.

Distributed Cloud Cache

In this section, we provide an overview of the proposed distributed cloud cache. The main idea is to have a distributed cloud cache that consists of several cache nodes distributed over the Internet and is shown in Figure 1. These cache nodes can be implemented using servers connected to the Internet or on fog platforms at the edges of the network. In addition, it is possible to use a combination of both at the same time. Each of the cache nodes has its own storage capacity, processing power, and network performance. In our caching technique, heterogeneous caches with different storage capacities, processing powers, and network performances can be easily utilized together to implement a distributed cloud cache.

The main function of the cache nodes is to hold parts of the Active Files and assist the data center in the download process to speed it up for the clients. An Active File is a file that has been recently actively downloaded by many users and there is a need to enhance the performance of downloading it. For example, this can be a just released software or game that is highly in demand, thus creating a huge number of download requests in a short period of time.

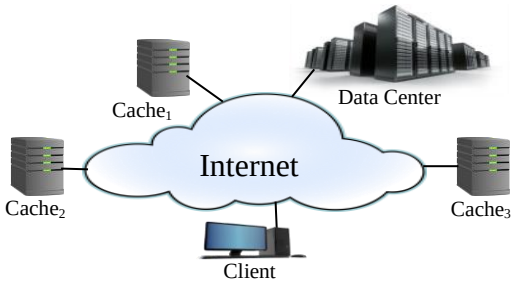

Figure 1: A distributed cloud cache

For best performance, the main functionality of the distributed cloud cache requires fully replicating the Active Files on the cache nodes if there is enough storage space. Otherwise, we can partially replicate the Active Files such that the aggregate result will generate at least a single full copy of the Active Files across all cache nodes and the cumulative download speed is maximized. As a distributed solution is used, it is better to utilize parallel delivery to clients to speed up the download. This significantly increases the download speed by avoiding two main constrains on the Internet and the cloud data centers:

- Avoiding network bottlenecks between the data center and the clients as the download process will be done in parallel for any client request. The data center and cache nodes are available in different places over the Internet. Thus, even if high network load exists close to the data center or some of the cache nodes, the other nodes can help in speeding up the download.

- The load on the cloud data center and the cache nodes can be very dynamic. Both the data center and the platforms used as cache nodes are not dedicated for the operations of the distributed cloud cache but will also have other functions. Consequently, it is difficult to know which of these nodes are unloaded to use for the download. However, utilizing an efficient parallel download process from the cloud data center and cache nodes to the clients can avoid such delays. In parallel transfer, the free nodes can download more parts of the requested Active Files than the busy nodes, yet the busy nodes will still provide some contribution in the process.

Unfortunately, as advantageous this approach is, there are several issues in implementing a distributed cloud cache:

- **Parallel Transfer Technique**: What is a suitable parallel transfer technique that can be used such that it can achieve good performance with the dynamic and best effort challenges of the Internet?
- **Cache Replacement Strategy**: As cache nodes have limited storage space, what cache replacement strategy will be suitable and efficient to replace blocks from inactive files with blocks from active files such that the accumulative download time is minimized?
- **Replication Strategy:** Replication of full active files enhances download times; however, it required large storage space on all nodes. Due to the limited storage space in most cache nodes, we need a good replication strategy such that a limited storage is used and at the same time good download time is achieved.
- **Other Management Issues**: As multiple cache nodes are used, we need a good model to manage the cache nodes and the file download requests.

In this paper, we propose solutions for these issues. We will first describe an efficient and fast parallel transfer technique for distributed cloud cache. Then, we will discuss the technique components and operations. Then, we will explain some efficient cache replacement and data pre-fetch strategies for the distributed cloud cache.

Fast Parallel Transfer

One of the main features of the proposed distributed cloud cache is the parallel transfer of the requested files from the data center and the cache nodes to the clients. For fast parallel transfer, we utilize DDFTP (Dual-Direction File Transfer Protocol) (Mohamed et al., 2013). However, in this paper, we use this technique with some modifications to suit the functions of the distributed cloud cache.

To start, we assume that we have full replications for all Active Files in all cache nodes. Although this assumption is not a realistic representation of the distributed cloud cache actual use model, we use it to simplify the explanation of the dual-direction parallel transfer technique. In Section V we will cover the more efficient (realistic) use model and see how to use the same technique with partial replication of the Active Files while maintaining good performance. Here, we will start with a simple case with one data center and a single cache node to show how the dual direction technique works, then extend the description to include multiple cache nodes.

One Cache Node:

Assume we have two complete copies of a file in both the data center and a cache node that is in a different location over the Internet relative to the data center. Based on the dual-direction parallel transfer technique, when a request arrives to download the file to a client, this file will be divided into several equal sized small blocks (e.g. 40 blocks in this example) as shown in Figure 2.

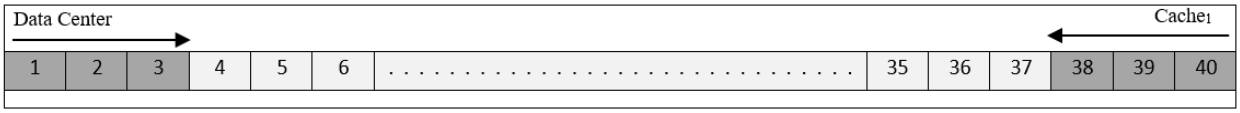

Figure 2: A file with 40 blocks, the data center started to download forward while Cache1 started to download backward.

The dual-direction parallel transfer is done by having the data center send the blocks forward starting from the first block from the left (beginning of the file) and moving right (i.e. sending blocks 1, 2, 3 …). At the

same time, the cache node starts to send blocks backward from the right (the end of the file) moving left (i.e. sending blocks 40, 39, 38 …). In the figure, the dark blocks are those already downloaded or on their way to the client, while the light blocks are those remaining (not sent yet). The client will receive the blocks from both the data center and the cache node and order them in the receiving file space. As the client will receive the blocks from both sides, it will ask them to stop sending more blocks as soon as it receives all file blocks.

Putting the client in control will allow the process to terminate as soon as the client has the full file and eliminate the need for the senders to keep track of what they sent. However, this will result in duplicate transfer of a few blocks by the two senders. These are necessary to improve the overall performance and will just be ignored by the client and will not significantly affect.

The flow of blocks will vary depending on the load on each side and the performance of the networks connecting them with the client. For example, the client can receive more blocks from the data center as shown in Figure 3 because:

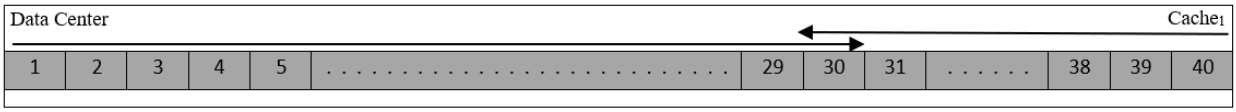

Figure 3: A case of the data center downnloaded more blocks comparing to Cache1 as the client is closer to the data center.

- The data center was free while the cache node was loaded with other tasks.
- The network paths between the data center and the client are unloaded while the network paths between the cache node and the client are loaded.
- And/or the client is closer to the data center compared to the cache node, thus there is a smaller number of hobs between the client and the data center compared to those between the client and the cache node.

However, the client can receive more blocks from the cache node instead as shown in Figure 4, due to many reasons:

- The data center is busy with other loads.
- The network paths between the data center and the client are loaded.
- And/or the client is closer to the Internet edge that has the cache node.

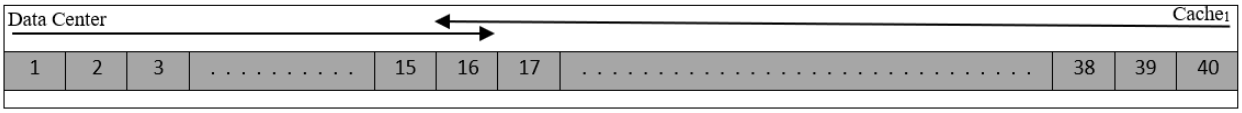

Figure 4: A case of Cache$_1$ downloaded more blocks comparing to the data center as Cache$_1$ is close to the client.

The dual-direction parallel transfer for a two-replica situation can achieve an optimal download time regardless of the load, network performance and location differences for the two senders. Simply put, each sender will provide best effort and the faster, better connected, or closer one will contribute more. However, the technique will guarantee that the complete file will arrive in the shortest possible time with maximum contributions from both sides.

Multiple Cache Nodes:

In the general case, we usually have more than one cache node available, thus it is important to see how the dual direction technique will work. Assume we have n cache nodes in a distributed cloud cache, where n is an odd number (for now). Let us also assume that we have full replication for an active file in all cache nodes. Therefore, there will be (n+1) replicas for that file in the cloud data center and in n cash nodes.

The active file will be virtually divided into (n+1)/2 equal parts and in each part, we have an equal number of small blocks. We will also divide the data center and cache nodes into (n+1)/2 pairs. The first pair will be the data center with cache node 1 (DataCenter, Cache1), while the second pair is Cache2 and Cache3, and so on. Each node pair will process one of the (n+1)/2 parts as if it is a single file (as explained in Subsection A.). Naturally, the load, network, and location conditions for the nodes in use are different from each other. As a result, after some time, some pairs will complete their part, while others are still sending their parts' blocks. Therefore, the pairs that finished move on to help other pairs that are still working on their parts. As an example, assume we have 3 cache nodes in a distributed cloud cache, Cache1, Cache2, and Cache3. Therefore, we will have 4 replicas of an active file. This file will be divided into 2 parts as shown in Figure 5 where each part has 20 equal blocks.

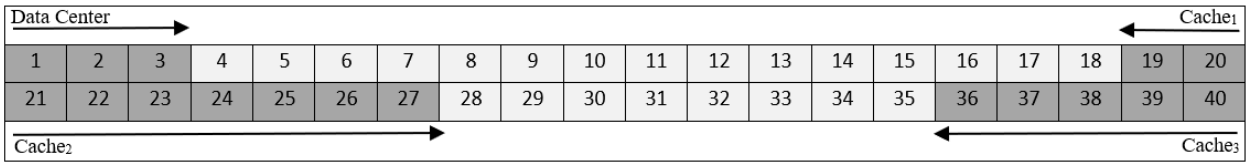

Figure 5: A case of blocks download are done by two pairs (DataCenter, Cache1) and (Cache2, Cache3).

The cache nodes and the data center will be divided into 2 pairs, (DataCenter, Cache1) and (Cache2, Cache3). We will have the first pair work on the first part of the file while the second pair work on the second part. After some time, we will have a situation such that one of the pairs (Cache2, Cache3) completed its part while the second (DataCenter, Cache3) is still working on its part as shown in Figure 6.

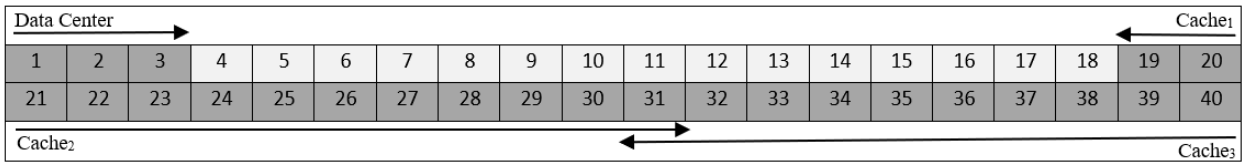

Figure 6: A case of the second pair (Cache2, Cache3) is done with its blocks.

Therefore, the (Cache2, Cache3) pair can help the other pair by creating a new pair assignment. This will result in having (DataCenter, Cache3) in one pair while we will have (Cache2, Cache1) in the second pair as shown in Figure 7.

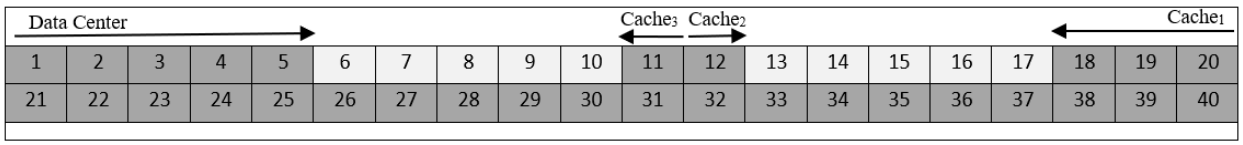

Figure 7: A case of a load-balancing step where Cache3 will help DataCenter and Cache2 will help Cache1.

The advantage of this assignment is that while the DataCenter and Cache1 are working on their tasks, Cache2 and Cache3 will start helping them without interrupting them. This type of reassignment will achieve good load balancing by letting all nodes work together to complete the remaining tasks without the need for prior arrangement or coordination. Unlike in our previous work in DDFTP where any server can download blocks from left-to-right sometimes and from right-to-left in other times, here, all cache nodes with odd IDs only download blocks from right to left while the data center and all cache nodes with even IDs only download blocks from left to right. So, if Cache2 receives a request to download from block 10, it will send blocks 10, then 11, then 12, etc. However, if Cache3 receives a request to download from block 10, it will send blocks 10, then 9, then 8, etc. This will reduce the management and implementation complications.

Generally, this dual-direction parallel transfer technique has very little overhead as there will be a small round of reassignments for achieving good load balancing even with large file sizes and with many replicas (Mohamed et al., 2013). More detailed information implementation, performance analysis, and advantages of the dual-direction parallel transfer can be found in (Mohamed et al., 2013).

Distributed Cloud Cache Components and Operations

In regular file download operations, the data center will receive requests for certain files from several clients. Upon receiving these requests, the data center directly starts to download the requested file. Utilizing a distributed cloud cache requires the use of a different data flow model to transfer the requested files from the data center and the cache nodes to the client. A software solution will be needed with components present at the data center, cache nodes and the clients. However, it is also possible to relieve the client and use other support nodes to complete the process without putting additional burden on the client.

Although, the main contributions of this paper are the concept of distributed cloud cache and efficient operations of this cache, we also discuss in this section a prototype implementation, which we later used for preliminary performance evaluation. In our approach we include several components: A Distributed Cloud Cache Manager (DCCM), Cache Node Managers (CNMs), and Client Proxies.

Distributed Cloud Cache Manager:

The Distributed Cloud Cache Manager (DCCM) is available at the data center and responsible for managing the overall operations of the distributed cloud cache. The DCCM maintains the list of cache nodes and their description including IP addresses and ports. In addition, the DCCM maintains a list of active files, their locations in the data center, and their sizes. Furthermore, it maintains information about where parts of each active file are replicated in the available cache nodes. The list of active files can be defined manually for the DCCM or automatically generated using different polices such as:

- A file is considered an active file if the number of requests to download it exceeds a specified threshold within a specified time frame.
- A file is considered an active file if the number of requests to download it is ranked among top x files in a list of y files within a certain time frame where x<y.
- A file can continue to be considered an active file if there is still some unused space in the cache nodes.

DCCM communicates with all cache nodes to let them know about adding or removing an active file. In addition, DCCM assigns, for each cache node, the staring block number that they will use to begin their

download for the clients. As we mentioned earlier, each cache node has an ID. If this ID is odd, the cache node works from right-to-left starting from the block number assigned, while other cache nodes (with even IDs) will work from the left-to-right also starting from the assigned block number.

Furthermore, the DCCM is responsible for receiving file download requests from the clients. With each request, the DCCM checks if the requested file is in the list of active files, it is not then it will use a normal way to download the file (i.e. directly from the data center to the client). Otherwise, it will initiate the fast dual-direction parallel transfer to transfer the requested file from the data center and the cache nodes to the client. To achieve this parallel transfer, the DCCM sends information about which cache nodes have parts of requested the file, where these cache nodes can be found, and from which block numbers they will start with to the client proxy.

Cache Node Managers:

Each cache node will be managed by a cache node manager (CNM). CNMs are responsible for receiving instructions from the DCCM to add or remove a file from the active files list. When removing an active file, the CNM will free the space used by the blocks of that file from the cache node storage. When adding an active file, the CNM will ask the DCCM to send the full or partial file from the data center based on the available storage space in the cache node. The full file will be requested and stored if there is enough space. Otherwise, only partial file will be requested and stored. In addition, the CNM can perform a cache replacement operation, if the storage is already full, to replace some infrequently downloaded blocks with new blocks from the new active file.

The CNMs are also responsible for receiving two types of requests from the clients. The first is requests to start downloading from a specific block number for a specific file. The second is to stop sending more blocks from the file. As a CNM downloads different blocks to different clients, it maintains information about the downloads including the time of last request received to download a given file and the number of downloads performed for each block since it was retrieved from the data center. This information is needed for the cache replacement strategy discussed in the next section.

Client Proxies:

Client proxies are available from the clients. A client can request a file and receive that requested file through its proxy. The client proxies are responsible for dealing with the DCCM at the data center and with the CNMs at the cache nodes. When a client needs a certain file, a request is sent from the client proxy to the DCCM. The DCCM can perform a regular download if the requested file is not an active file; otherwise, the DCCM sends information to the client proxy including a list of cache nodes and their information in addition to the parts of the file that can be obtained from each cache node. The client proxy is responsible for requesting the parts of the file from the corresponding cache nodes and specifying the starting block number to start downloading blocks to the client. The client proxy will receive the blocks and put them in their correct places in the receiving file space. Moreover, the client proxy implements the dual direction load balancing mechanism mentioned in the previous section. The client proxy can ask the cache nodes to stop downloading more blocks from the current position and restart downloading from another block in the file to achieve load-balancing and, as a result, a good download performance.

Cache Replacement and Pre-Fetch Strategies

Having several large active files, there will be no space to replicate all of them on all available cache nodes as most of these nodes will have limited storage capacities. Therefore, a distributed cloud cache should

partially distribute these files in the available spaces in the cache nodes such that the overall performance will not be negatively affected. At the same time, the list of active files can be changed periodically based on demand and access. In addition, it may be possible to have different active file lists in different areas across the Internet. For example, an active file may be requested in one city more than other cities. Therefore, there is no sense to have the same full replication or the same partial replication in all nodes of the distributed cloud cache. All these issues require good cache replacement and partial replication strategies to enhance the overall download performance of active files and optimize storage space utilization at the cache nodes.

Cache Replacement Policy:

A cache replacement strategy is needed to be part of the CNM to define which currently available block from any of the active files in the cache node should be replaced with another block from the same or another active file to enhance the overall download performance. We developed the following policy for this replacement. Assume we have a file with 100 blocks, and we only have one cache node. While with one cache node, we will not have a distributed cache, it is enough to explain our strategy that can easily and directly be applied when multiple cache nodes are used. Furthermore, to simplify the description, also assume that the file is completely replicated in Cache1.

Figure 8 shows the download statistics after 100 downloads of the file from both the data center and Cache1 using the dual-direction parallel transfer. The figure shows that blocks 74 through 100 were never downloaded from the data center, while blocks 1 through 40 were never downloaded from Cache1. Therefore, as blocks 1 to 40 are never downloaded from Cache1, then there is no sense to have them stored in Cache1 and CNM can replace them with other more important blocks. As these blocks are removed, new requests for the same file will still be satisfied in the exact same way, thus there is no negative impact on the download performance. The only case where an impact will occur is when the data center crashes for some reason and the cache node ends up having to deliver blocks 1 through 40, which it does not have anymore. However, in a real distributed cloud cache, there will be multiple cache nodes, which collectively will own a complete copy of active files in some organization and the needed blocks will be delivered eventually by other nodes.

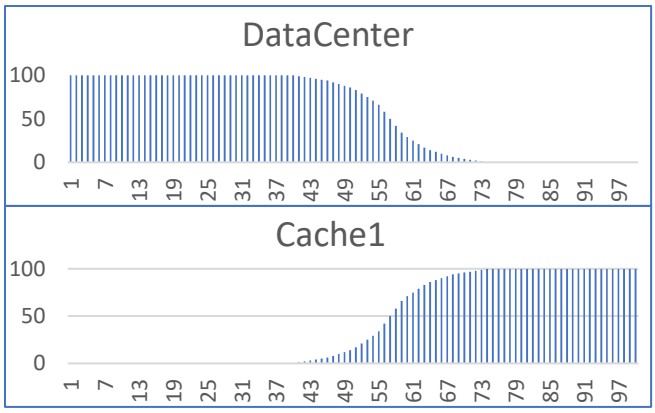

Figure 8: Download statistics after 100 downloads, the vertical-axis is the number of downloads while the horizontal-axis is the block number in the file.

Now, if all blocks with zero download are removed from the cache while still there is no space to hold more important and new blocks, then it easy to identify which blocks from the cache should be removed. As we can see from Figure 8, blocks 41 to 50 are downloaded a few times from Cache1. Therefore, CNM can

replace them with others. Although, the download performance time may be affected slightly. This is not much as these blocks are downloaded very few times out of 100 downloads. As a result, removing these blocks will only have a little impact on the download performance time.

The cache nodes in distributed cloud cache will have multiple active files at the same time. The issue is that if there is no space while a replacement is needed for new blocks and at the same time there are multiple available blocks for different files with zero or little downloads, then which block should be selected to be replaced? Although, different policies can be used, we use the following two strategies:

- If there are multiple active files, then replace blocks with zero or minimum downloads from the least recently downloaded active file.
- Then, if there are few blocks to be removed from that active file, blocks with less chances to be downloaded in the near future will be selected for replacement.

We developed algorithms for both strategies; however, we will not discuss them in this paper due to the space limitations. When we have multiple cache nodes, each node will make independent decisions to replace any blocks it has using the same methods. Each cache node will make these decisions based on its situation, currently active files, available space, and recent download patterns.

Blocks Pre-Fetch Strategy:

As we have seen in the previous subsection, there are usually some blocks that will never be used in each cache node. Therefore, even without full replication across all cache nodes, we can use the dual-direction parallel transfer and still achieve the best performance possible. It is enough to have only some of the blocks of an active file in each cache node such that collectively, the cache nodes will have one or more full copy of the active files for efficient dual direction downloads. To ensure proper coverage we start by providing each cache node with a few blocks of an active file being requested. These blocks are defined by the DCCM. As a cache node starts to send the blocks (it already has) from the requested active file to the client, it can also pre-fetch more blocks from the data center as needed.

For example, Assume Cache2 already has a copy of blocks 60 through 70 and it starts to send blocks 60, 61, etc. to the client. Cache2 can now request blocks 71 onwards from the DCCM, which will push these blocks to Cache2 from the data center. This pre-fetch helps ensure the blocks are available if Cache2 needs to send more blocks than it originally had. The newly downloaded blocks from the data center may be sent to the client but will also be kept in Cache2 for possible future use. As all cache nodes will apply this mechanism then they will all have the needed blocks to maintain good download performance for their clients.

Implementation

A prototype for the proposed distributed cloud cache was developed, where the DCCM, CNM, and client proxy were built in Java. The DCCM and CNM are servers that provide services for clients while the proxy acts as a client that can utilize the DCCM and CNM services. We linked the DCCM, multiple CNM, and multiple client proxies using MidCloud (Mohamed and Al-Jaroodi, 2015). MidCloud is an agent-based middleware we developed previously to provide cloud clients with dynamic load balancing and fault tolerance when utilizing replicated cloud services. Although, we may not have full replication on the data center and all cache nodes as we rely on partial replication with distributed cloud cache, the data can be viewed as fully replicated everywhere from the clients' perspective. Partial replication and partition management will be performed transparent to the clients. MidCloud can deal with two types of services:

data services and computation services. We only used data services and defined both DCCM and CNM as they provide data services. In addition, we did not use the fault tolerance feature of MidCloud but only the load balancing feature. The load balancing feature is implemented using the dual-direction parallel transfer technique but without including the mechanisms for distributed cloud caching we propose in this paper. These mechanisms were newly implemented by both DCCM and CNM. DCCM was implemented to manage all cache nodes through their CNMs. DCCM has data retrieval services that can be used by both CNMs and client proxies while CNMs have data services that can be accessed by the client requests. We also implemented in the CNM the cache replacement and pre-fetch strategies.

Performance Evaluation

To evaluate the performance gains of the proposed distributed cloud cache, several sets of experiments were conducted using the prototype implementation. We used several computers to represent a data center, cache nodes, and clients. In addition, we used a wide area network (WAN) emulator to connect these computers to include the effects of WAN such as high return trip time (RTT), limited bandwidth, and packet drops. The first set of experiments was conducted to compare the performance of file downloads using the proposed distributed cloud cache solution with the regular FTP using only the data center.

In this set of experiments, we used the distributed cloud cache with one cache node (DCC1), three cache nodes (DCC3), and five cache nodes (DCC5). The performance of the regular FTP, DCC1, DCC3, and DCC5 were measured and compared for downloading five different files of sizes 100MB, 200MB, 300MB, 400MB, and 500MB. For this set of experiments, we configured all cache nodes to have enough space to hold all files. Furthermore, a block of size 4000 bytes was used for all experiments. Each file was downloaded 100 times, and the average time of all downloads was taken. Figure 9 shows the performance gains of the distributed cloud cache solution over the regular FTP. As shown in the figure, the performance gain of distributed cloud cache is generally good, and it also increases as the files grow larger. In addition, as we use more cache nodes, the performance gain is further enhanced.

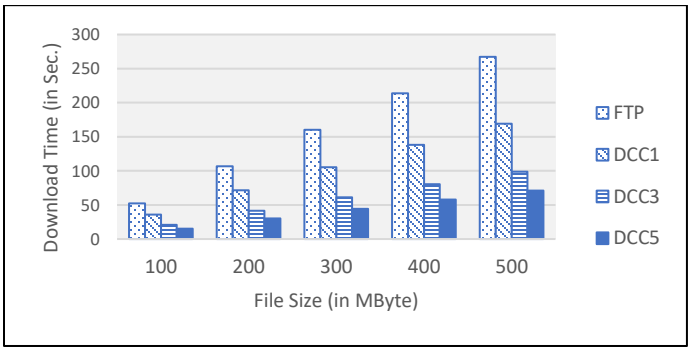

Figure 9: FTP Vs. DCC1, DCC3, and DCC5 with heterogenous load on the data center.

The second set of experiments was conducted to study the impact of limited size caches on a file download performance. We used a three-node distributed cloud cache (DCC3) and a file of size 400MB. We limited the space in each cache node to 400MB, 300MB, 200MB, 100MB, 80MB, 60MB, and 40MB. The file was downloaded 100 times and the average time of all downloads was taken, see Figure 10.

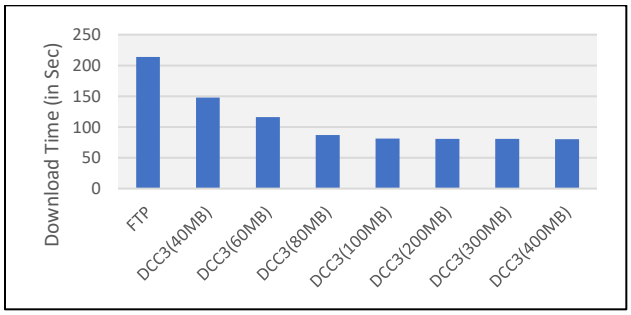

Figure 10: Impact of different cache sizes on the performance of downloading a 400MB file. 3 cache nodes were used.

As we can see from the figure, the performance gain can be obtained even with small caches such as 40MB compared to the regular FTP used through the data center alone. The performance gain can be enhanced with larger storage space in the cache nodes. However, no more performance gain can be obtained as we increase the size beyond 100MB. With a file of size 40MB, on average, each participating node will have to send a quarter of this file to the client. As a result, each will only need to keep that quarter of the file. Any additional storage space will not add to the performance gain.

Therefore, there is no need to have a full replication of the file to obtain good performance. The overall performance may be affected slightly if one of the nodes becomes a lot slower than the others, thus requiring more blocks to be downloaded from the other nodes. In this case the cache replacement strategy will be used, if we have some unused blocks from other files, or the datacenter and other cache nodes will take on the responsibility of delivering the remaining blocks. This will introduce a small additional overhead, while still performing better than a traditional FTP download from the data center.

The third set of experiments was conducted to measure the performance gain of using the distributed cloud cache over the regular FTP when multiple clients concurrently download the same file of size 100MB. In this set of experiments, we used 6 clients and a distributed cloud cache with 3 cache nodes (DCC3). Each cache node has 30MB storage space. We used the WAN emulator to limit the aggregate bandwidth available from the data center to these 6 clients to 1, 2, 3, 4, and 5 MByte/Second. This is to emulate the existence of other load on the data center, where at 1MByte/Second the data center will be very busy compared to 6MByte/second. We also limited the available bandwidth that each cache node can provide to all clients to 1MByte/Second for all experiments. We compared the performance gains of downloads using distributed cloud cache against FTP under different cloud loads. Figure 11 shows the experiments' results, where we see the performance gain is very high when the data center and its network connections to the Internet are very loaded.

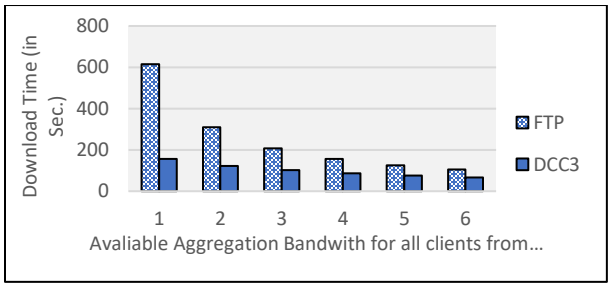

Figure 11: Multiple file downloads while the data center and its network connect to the Internet are loaded.

The fourth set of experiments was conducted to measure the performance gain of using the distributed cloud cache against utilizing a single cache solution as in (Han et al., 2011; Kantere et al., 2011; ElastiCache, 2024). In the single cache solution, any download request is satisfied by using the closet cache to the client. The data center will be used for the file download if it is the closest to the client compared to the available caches. Four clients were used. The testing environment was configured to have different effective bandwidths between the data center/caches and the clients as shown in Table 1. Figure 12 shows the results of both solutions showing a significant improvement using the proposed distributed cloud cache for file with size 200MB.

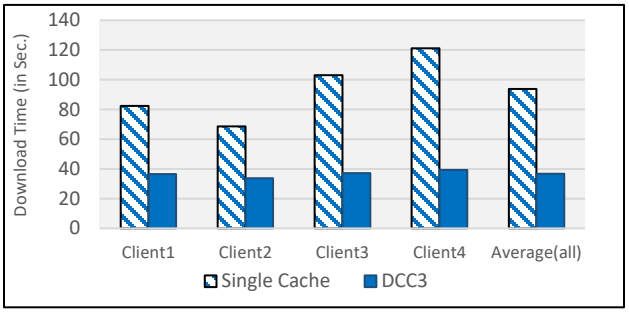

Figure 12: The performance of distributed cache solution against the single cache solution.

On average the time needed to download a file in the proposed technique is around 39% of the time needed to download the file by using a single cache. In addition, the single cache solution required having a full replica of the file, that is 200MB, in each cache node used (i.e. using 600MB of space across three cache nodes). In our approach, each cache node managed to support all clients' requests with partial replicas where each cache node had to keep an average of 90MB of the file thus needing a total of 270MB of space. As a result, the proposed solution does not only enhance file download time, but it also reduces the needed space for replication without compromising the overall download performance.

Table 1: Effective bandwidth between different clients and the data center & caches (MByte/Sec.)

Client #	DataCenter	Cache1	Cache2	Cache3
Client1	2.5	1.5	1.2	0.6
Client2	1.5	3.0	1.0	0.8
Client3	1.3	1.4	2.0	1.0
Client4	1.1	1.2	1.4	1.7

Using partial replication, the partial copies are automatically formed based on the available space, current load, defined active files, location and number of the used cache nodes, and the number and requests from clients.

Conclusion

The use of cache has been proven to improve performance in various situations where there is a mismatch in performance between different system components. The concept has been applied in memory hierarchy, Internet caching, and storage caching as well. In this paper, we extend the concept to create an efficient and highly optimized download performance for clients using distributed cloud cache. Our proposed technique extends the traditional cache method by distributing the download process across multiple cache nodes instead of relying on a single cache node to do all the work. Thus, when a large data set or file is requested, the data center can enlist all available cache nodes to help download different parts of the file in parallel.

The client will require the use of a proxy that will control the download process and rearrange the file blocks as they arrive at the client site. As the experiments demonstrated, we managed to achieve very good performance for the downloads compared to the use of the data center alone or the use of the single closet cache node to the client.

Another aspect where we achieved improvement is eliminating the need for full replication of the files across all cache nodes used. We developed a partial replication method, where only parts of the files are stored in each cache in a way that allows each one to fully participate in the download process; while not having to keep the parts it does not use in that process. On average, we could reduce the cumulative storage space needed across all cache nodes to far less than half of that needed if all cache nodes carried a full replica of the files. With mode cache nodes in use, more space can be saved as each node will need to keep smaller parts of the files.

Distributed cloud cache significantly improved the download performance and the storage space utilizations. The experimental results demonstrated those gains. However, this technique can further be improved and optimized. We plan to further investigate different cache replacement algorithms to enhance that part of the process. We also plan to dedicate more effort in enhancing the download monitoring to obtain a better view of how the process can be further optimized based on the analysis of different download behaviors and patterns over time. Moreover, there is always room to investigate how this approach can be integrated with other technologies and applications to enhance overall cloud-based processes and services that require data transfers as part of their overall workloads.

References

Abdalla, H. B. (2022). A brief survey on big data: technologies, terminologies and data-intensive applications. *Journal of Big Data*, *9*(1), 107.

Amazon ElastiCache, URL: https://aws.amazon.com/elasticache/, viewed May 2024.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, 53(4), 50-58.

Arteaga, D., Otstott, D., & Zhao, M. (2012, February). Dynamic block-level cache management for cloud computing systems. In *Conference on File and Storage Technologies*.

CERN: The European Council for Nuclear Research, viewed May 2024, URL: http://home.cern/about

Han, H., Lee, Y. C., Shin, W., Jung, H., Yeom, H. Y., & Zomaya, A. Y. (2011). Cashing in on the Cache in the Cloud. *IEEE Transactions on Parallel and Distributed Systems*, *23*(8), 1387-1399.

Kantere, V., Dash, D., Francois, G., Kyriakopoulou, S., & Ailamaki, A. (2011). Optimal service pricing for a cloud cache. *IEEE Transactions on Knowledge and Data Engineering*, *23*(9), 1345-1358.

Liu, S., Huang, X., Fu, H., & Yang, G. (2013, May). Understanding data characteristics and access patterns in a cloud storage system. In *2013 13th IEEE/ACM International Symposium on Cluster,*

Cloud, and Grid Computing (pp. 327-334). IEEE.

Makuch, E. (2015) "Call of Duty: Black Ops 3 Generates $550 Million in Three Days," GameSpot, viewed May 2024, URL: http://www.gamespot.com/articles/call-of-duty-black-ops-3-generates-550-million-in-/1100-6432185/

Memchached, URL: http://www.memcached.org/, viewed May 2024.

Mohamed, N., & Al-Jaroodi, J. (2015). MidCloud: an agent-based middleware for effective utilization of replicated Cloud services. *Software: Practice and Experience*, *45*(3), 343-363.

Mohamed, N., Al-Jaroodi, J., & Eid, A. (2013). A dual-direction technique for fast file downloads with dynamic load balancing in the cloud. Journal of Network and Computer Applications, 36(4), 1116-1130.

Redis, URL: https://redis.io/, viewed May 2024.

Shen, M., Kshemkalyani, A. D., & Hsu, T. Y. (2015, May). Causal consistency for geo-replicated cloud storage under partial replication. In *2015 IEEE international parallel and distributed processing symposium workshop* (pp. 509-518). IEEE.

Williams, D. N., Ananthakrishnan, R., Bernholdt, D. E., Bharathi, S., Brown, D., Chen, M., Schwidder, J. Bharathi, S. & Wilhelmi, N. (2009). The Earth System Grid: Enabling access to multimodel climate simulation data. *Bulletin of the American Meteorological Society*, *90*(2), 195-206.

Wu, Z. (2017). *Cost-Effective Support for Low Latency Cloud Storage* (Doctoral dissertation).

Zhang, J., Wu, G., Hu, X., & Wu, X. (2012, September). A distributed cache for hadoop distributed file system in real-time cloud services. In *2012 ACM/IEEE 13th International Conference on Grid Computing* (pp. 12-21). IEEE.