Adapting InfiniCache to Support Deep Learning Framework

Yu Han, *George Mason University*

Abstract

Recently a published paper presents INFINICACHE [1], a prototype of a serverless distributed memory caching system deployed atop AWS Lambda. The benefits of using InfiniCache instead AWS ElastiCache are that on the client side, the users can achieve 31 – 96x cost savings for a 10 MB-file only production job, it can effectively offer 95.4% data availability per hour with a relatively comparative performance compared to other typical in-memory caching system. Meanwhile, another published paper introduces Quiver [2], an informed storage cache for deep learning training jobs in a cluster of GPUs. By codesigning with the PyTorch [12], Quiver's team develops a technique of substitutable cache hits algorithm to increase its caching capacity in order to get more values as a storage cache and also to avoid cache thrashing when the working set is much larger than the capacity of the storage cache, therefore, the team shows that Quiver can significantly increase the effectiveness of deep learning training jobs. It provides a great alternative storage solution instead a traditional way staging of data to local SSD manually. The main idea of this project is to implement a Python-based client for InfiniCache as the original source code is written by the Golang. The client is designed to port he InfiniCache into an existing deep learning framework, in this project which is PyTorch. The intention of this project will be learning the serverless computing (AWS Lambda), understanding the system of InfiniCache and adapting InfiniCache to support deep learning frameworks (PyTorch), ideally, the deep learning training jobs can use InfiniCache as the caching system which can have a significant cost savings for deep learning training jobs and it can be used as the reference for further implementation of LambdaQuiver which is needed two additional steps. Firstly, implement controlled probing (substitutable cache hit mechanism). Secondly, make sure the controlled probing and Python-based client can work in a multi-job environment where the multiple deep learning training jobs share the LambdaQuiver cache.

1. Motivation

Deep learning has been widely used for many of the recent successful achievements such as AlphaGo, autonomous driving cars (Tesla) and intelligent voice assistants (Siri). Deep learning frameworks are speeded up with graphics processing unit, the deep learning training jobs can be accelerated from most few weeks to least few hours. Deep learning frameworks provides building blocks for designing, training, and verifying deep neural networks through advanced programming interfaces. Every major deep learning framework such as Caffe2, Chainer, Microsoft Cognitive Toolkit, MxNet, PaddlePaddle, TensorFlow and PyTorch relies on the deep learning SDK library to provide high performance multi-GPU accelerated training. However, the storage layer of deep learning training jobs has not been improved much for a period. Quiver [2] is introduced to highly improve the throughput of deep learning jobs. The Quiver is implemented on a cluster of 48 GPUs across 12 virtual machines on Microsoft Azure. 6 of them contain 4 Nvidia P100 GPUs each and the other 6 contain 4 Nvidia P40 GPUs each. All virtual machines contain 3 TB of local SSD. Therefore, there comes a question: "How much does this cluster cost for running a month?" as most distributed caching storage run on a cluster of virtual machines that are required to run 24/7, the cost of a running cluster is fixed and depends on required memory allocation. As for the experimental setup in Quiver, a single virtual machine running 730 hours cost $2,193.85, if there are not many requests to the server, the costs are not effective. InfiniCache, a first-of-its-kind serverless pay-per-use caching system can improve the cost savings part without significantly reduce the performance compared to the typical fixed pricing in-memory caching system. Ideally, the price of running the exact same virtual machine 730 hours on Microsoft Azure can as low as $22.85. The motivation of this project is to port the InfiniCache into the deep learning frameworks (PyTorch) to substitute the storage layer of deep learning training jobs. I worked on a deep learning training job of training models using GoogleNet. A typical training dataset is the ImageNet large scale visual recognition challenge 2012 (ILSVRC2012) [4] which is about 100 GB and it includes 1,000,000 images with more than 1,000 classes. InfiniCache is designed to fit this scenario to handle large object with comparative performance and much less cost.

1.1. Problem statement

Typical distributed caching system performs not well when it handles large objects as small-size objects are accessed more often while requiring immense space in the random access memory (RAM), access to the large

objects requires a vast amount of network bandwidth to cause a lag for accessing the small objects. In this case, Quiver is introduced to support the I/O bandwidth requirements of deep learning training jobs to significantly improve cache efficacy. Having InfiniCache to only serve web applications' I/O workloads is far from enough. Adapting InfiniCache to cache the I/O workloads of deep learning training applications. Quiver, an informed storage cache for deep learning training applications, which will be built atop Lambda-boosted InfiniCache. The idea of project is to port the InfiniCache to support the deep learning framework with the characteristics of Quiver that combine the advantages to benefit the deep learning training applications.

2. Approaches

The InfiniCache's client library is needed as the Lambda [14] node is not able to run in server mode therefore, the client must connect to an intermediate server which is the function of the EC2 [11] proxy. The client library is responsible to handle the object encoding and decoding using an embedded EC module, keep the load of the requests among all proxies balanced and determine the destination of the EC-encoded chunks on a cluster of Lambda nodes. To implement a Python-based client library that handles the API of the EC2 proxy that is compatible with the PyTorch.

2.1 Experimental setups so far

InfiniCache contains three parts: an InfiniCache client library, an EC2 proxy and a Lambda Runtime used to implement cache nodes. I have set the EC2 proxy which is used to manage a bulk of Lambda nodes and transmit data between the client and Lambda nodes. And I have created an AWS S3 [10] bucket for storing the zip files of the Lambda code and data output generated by the Lambda functions but not yet configured the AWS credential to bond the S3 bucket and the Lambda function. The next step will be setting up the Lambda Runtime and configuring the Lambda execution in order to deploy the Lambda functions.

3. Expected milestones

Firstly, Setup the environments of InfiniCache. Secondly, study the source code of the EC2 proxy to understand the work system. Thirdly, study the InfiniCache Client Library. Finally, implement a Python-based Client Library as the original source code is written in Golang then port the InfiniCache to the Python-based Client and integrated with the deep learning framework (PyTorch)

4. Related work

INFINICACHE: Exploiting Ephemeral Serverless Functions to Build a Cost-Effective Memory Cache [1]. The paper presents the prototype of InfiniCache which is built atop a popular serverless computing platform (AWS Lambda), it has a feature of pay-per-use to significantly reduce the cost by 31 x compared to Amazon ElastiCache while it is able to keep 95.4% of data availability per hour.

Quiver: An Informed Storage Cache for Deep Learning [2]. This paper introduces Quiver which is an automated storage cache codesigned with PyTorch to significantly improve throughput of deep learning workloads.

RedisAI - A Redis module for serving Deep learning and Machine learning models.[6] RedisAI is a Redis module for serving tensors and executing deep learning models. Redis is an open source, in-memory data structure store, used as a database, cache and message broker. InfiniCache also serves as a caching system just like Redis, this can be used as a helpful reference to co-design the InfiniCache with PyTorch.

References

[1] InfiniCache: Exploiting Ephemeral Serverless Functions to Build a Cost-Effective Memory Cache

https://www.usenix.org/conference/fast20/presentation/wang-ao

[2] Quiver: An Informed Storage Cache for Deep Learning
https://www.usenix.org/conference/fast20/presentation/kumar

[3] InfiniCache: Distributed Cache on Top of AWS Lambda (paper review)
https://mikhail.io/2020/03/infinicache-distributed-cache-on-aws-lambda/

[4] InfiniCache GitHub repository
https://github.com/mason-leap-lab/infinicache

[5] Nvidia Deep Learning

https://developer.nvidia.com/deep-learning

[6] RedisAI **https://oss.redislabs.com/redisai/**

[7] RedisAI GitHub repository
https://github.com/RedisAI/RedisAI/

[8] Amazon ElastiCache

https://aws.amazon.com/elasticache/

[9] Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker https://redis.io/

[10] Amazon Simple Storage Service (Amazon S3)
https://aws.amazon.com/s3/

[11] Amazon Elastic Compute Cloud (Amazon EC2)

https://aws.amazon.com/ec2/

[12] PyTorch https://pytorch.org/

[13] A Berkeley View on Serverless Computing.

https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.pdf

[14] AWS Lambda https://aws.amazon.com/lambda/

[15] ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)

http://www.image-net.org/challenges/LSVRC/2012/