An Implementation of Mapreduce based on Dynamo

Yangyang Zhang, Xiao Mao
Department of Computer Science
University of San Francisco
{yzhang1117, xmao4}@usfca.edu

Abstract

Nowadays, technology companies need to process huge datasets every day. Large scale data processing has become a challenge. This challenge could be broken down to two parts: highly available and consistent distributed storage system; a parallel programming model for distributed system. High availability at massive scale is one of the biggest problems that e-commerce company has to tackle. Amazon, one of the biggest e-commerce company implements a highly key-value store storage system named Dynamo, which promised to provide an always-on experience. MapReduce, which is able to process large data sets in parallel in distributed system, is an efficient programming model to large datasets.

In our project, we combine Dynamo and MapReduce techniques to process data. Dynamo is our underlying data storage system, and Mapreduce is used to process large data sets stored on Dynamo. This paper presents implementation Dynamo based on pynamo and Mapreduce applications which run on Dynamo server.

1 Introduction

Amazon is one of the biggest e-commerce company in the world. It needs to serve large amounts of user requests each day. User experience is the key factor that Amazon needs to take into consideration. In order to gain good user experience, Amazon needs to guarantee reliability and scaling needs. To tackle these problems, Amazon has developed Dynamo [2]. Dynamo is a proprietary, highly available key-value a distributed key-value data store. Based on trusted infrastructure, Dynamo is always writeable and do not support for hierarchical namespaces and complex relational schema. To achieve high availability, Dynamo sacrifices a little consistency. It only provides eventually-consistent storage. Dynamo is highly available and provides desired levels of availability and performance, which is been shown in real environment.

In project 3, Dynamo is our basic file system, and our MapReduce [1] application is built on Dynamo. Part A of this project is to successfully run our version of Dynamo on distributed system. For this part, we decide not to implement Dynamo from scratch. There is a good implementation of Dynamo written in python named pynamo [5]. The flaws in pynamo is that it does not have any network communication. It passes messages by network simulation. In addition, pynamo keeps data in memory and does not support any data persistence. In this case, network communication has been added to pynamo by using Zerorpc [6], which is a light-weight, reliable and language-agnostic library for distributed communication between server-side processes, and data is stored in leveldb [3], which is a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.

The goal of part B of this project is Extending Pynamo. And our group chooses to implement MapReduce [1] based on pynamo. MapReduce is a programming model which is able to process and generate large data sets. This model is easy to use, even for programmers who are not proficient at parallel programming, because the model is responsible for scheduling the program's execution, fault-tolerance, locality optimization, load balance and dealing with machine failures. In this model, after user program calls the MapReduce function, the MapReduce library first splits the input files into pieces. Then the master assigns work to each worker. Each worker reads the contents of the related input split and passes key/value pair to Map function. The intermediate key/value pairs generated by Map function, which are buffered in memory, will be written to local disk periodically. And the location information will be passed back to master. When a reduce worker receive the location message, it go through the sorted intermediate data then passes the key and the corresponding values to users Reduce function. When all map and reduce have been completed, the whole operation completed. And we will discuss the implementation details of MapReduce in Section 2.

This paper describes the design and implementation of Mapreduce which is built on pynamo. The rest of this paper is organized as follows. Section 2 details the implementation. Section 3 presents the results of some basic experimentation of mapreduce applications. Section 4 makes some concluding remarks.

2 Implementation

This section describes the key implementation details to support MapReduce based on pynamo. This project is divided into two parts: modification on pynamo and MapReduce. We will first introduce how to make pynamo work in a real distributed system environment with network layer and then introduce the whole process of MapReduce, which is the essential part of this project.

2.1 Modified Pynamo

In part A, there are several issues needed to address such as message passing, object serialization and data persistence. And we choose Zerorpc for message passing, pickle for object serialization and Leveldb for data persistence. The above three techniques are a relative simple way to get pynamo work. The original version of pyanmo is well organized, so we decide to keep the original code as many as possible rather than writing our own code to avoid creating new bugs.

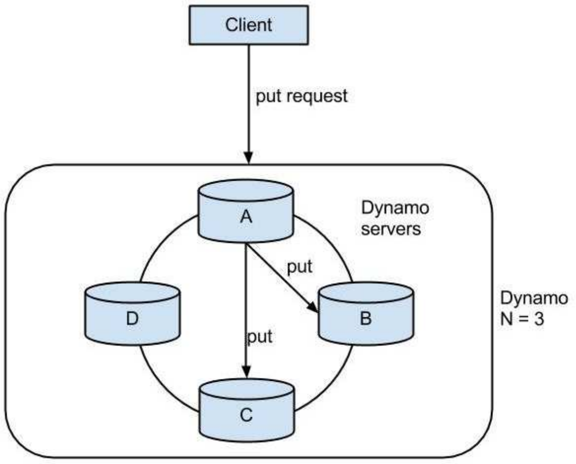

Figure 1. An Example of Put Operation

In original version of pynamo, framework. py is the file used to simulate network. Therefore, our work is to modify this file to realize actual message passing. We add Zerorpc connection for each dynamo server in the distributed system. Each time when one dynamo server tries to sends message, it passes a Message object and a connection to the Framework. send_message() function. Since the original dynamo encapsulates message information into a Message object, it is necessary to use pickle module to dump the Message object before sending message, and the receiver needs to use pickle module to loads the Message object. To store data in our file system, each time store() is called, data will be stored in Leveldb locally. Figure 1 is a simple example of a put operation. Assume that there are 4 dynamo servers running and the replica factor is 3. First of all, the client side sends put request to dynamo server. Then dynamo server will generate a preference list, for instance A,B,C. Node A store the key value pair locally and sends put request to node B and node C. After successfully storing data in Leveldb, dynamo server will return a list of node address where the key value pair is stored. And MapReduce master will keep this list to intelligent assign jobs to the workers.

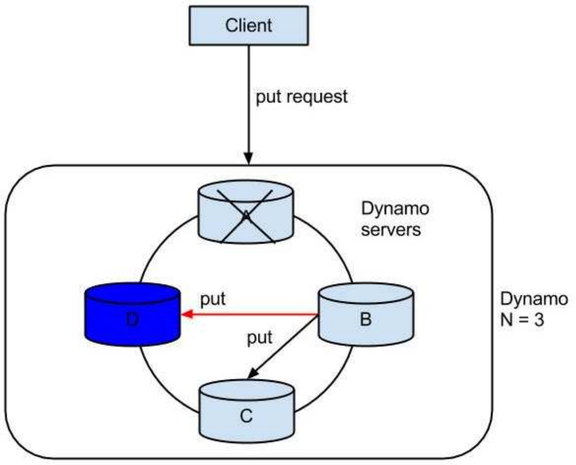

Figure 2. Node Failure

Moreover, server failure is a commonplace in cluster. That is another problem needs to handle. The basic idea is that if a dynamo server crashed, then the first node after the nodes in the preference list will store data for the failed dynamo server and it will periodically check the failed dynamo server. If the failed dynamo server is up again, it sends the hinted handoff data to the recovered dynamo server. Figure 2 shows that node A crashed, then those key value pairs which belong to node A will be stored in node D (a dictionary pending_handoff, key is the failed node and value is the keys belong to this node), the first node after the nodes in the preference list which is D. In this case, node D periodically sends retry message to the failed node A. If node A responds to this retry message, D knows that A comes back online. Then D sends put requests to A to store the associated key value pairs belong to A and deletes keys in pending_handoff. And that is the recovery process. And since version control is not our major concern in this project, so we does not implement merkle tree.

2.2 MapReduce

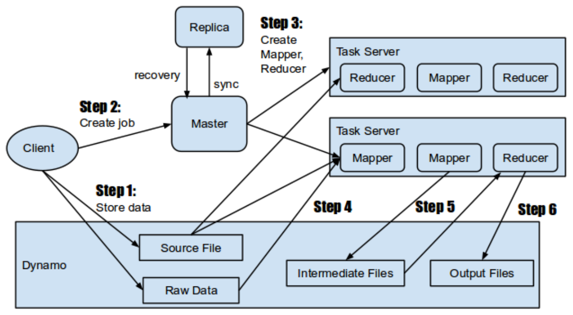

Figure 3. Overview

MapReduce is a programming model for solving parallelizable problems across big datasets. A MapReduce system usually consists of a huge number of commodity computers, and bas on a distributed file system. It was explained and used by Google [1], and then got more different implementations, Apache Hadoop is a popular free one. We refer to Hadoop, implemented our own MapReduce system based on Pynamo.

2.2.1 Workflow

There are three kinds of servers in the MapReduce system:Pynamo server, master server and task server.Pynamo servers are used as distributed storage. master server is responsible for scheduling and managing all user-defined jobs.There is only one master server in the system, and it can have some replicas in case of failure. Task servers are hosts of virtual workders:mappers and reducers, which are controlled by master server.

Each user-defined job is excuted as the follow steps:

1. Put a source file into Pynamo system. The source file should contain a MyMapper class that inherited from Mapper and a MyReducer class that inherited from Reducer. Both of them are defined by user.

2. Put data that are going to be processed into Pynamo system.

3. Create a Job with a unique name by calling new_job() function on master server.

4. Run the Job by calling run_job() function on master server.

These are all that users need to do for execute a new job. The following steps are automatically executed by servers:

1. To run a new job, master server creates some mappers and reducers on task servers at first. Input list are separated and send to mappers respectively. Mappers and reducers are treated as general workers in task server, and the amount of mappers and reducers for each job can be set by user. We create reducers before the job starts, so that if n reducers are assigned to the job, mappers of the job will partition their outputs into n pieces and store each piece as a separate file in Pynamo.

2. Master server notify all the mappers of the job to start. When a mapper receive the notification, it firstly read the source file and input data from Pynamo, then dynamically call map function of MyMapper class with input data as parameter. After finishing the map function, mappers pickle the output data and store the pickled data into pynamo as intermediate files, and return a list of file names to master.

3. Master is responsible for tracking how many mappers have done their work. When All mappers of a job are finished, it will come to the reducing stage. Firstly, master picks nth file name in each list that mappers sent to it and store them in another list, then the list is send to nth reducer. Each reducer read source file and all the files from Pynamo with file names in the list, and unpickle the data, then call the reduce function of MyReducer class multiple times, with one unpickled file as parameter each time. When all the data is processed, output function of MyReducer class will be called, output data be stored in Pynamo, and then the key of output is returned to master server. When all the reducer of the job finish their work and notify master, master marks the job as completed. Users can get the status of the job they created by querying the master with the unique name of the job as parameter.

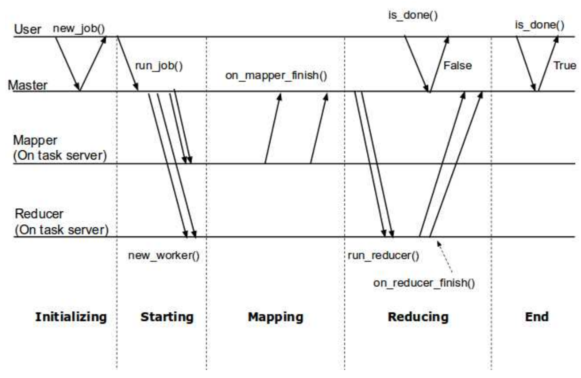

Figure 4. Workflow

2.2.2 Command Line Tool

The system is to run programs that are written by users, we can make sure there are no deadly bugs in our code by carefully checking the code of our system, but we are impossible to do the same thing to users' code. If a job was created and started, then the user who wrote it found that the program incorrect and would not produce right result, then the resources held by the job would not be released until the the job is done. Even worse, some users may write programs which contain infinite loop, and the resources held by them will never be used by other jobs. Given these are likely to happen, we provide a command line tool to manually kill a running job.

3 Experiments

To demonstrate the usability of the MapReduce system we have performed some simple experiments such as word counting and face recognition. We managed to run a face recognition program in the system, which is basically based on OpenCV [4].

The input data for this test is a bunch of raw photos and a target portrait.The map function is to find all faces in each photo and then crop and store those parts of photos into Pynamo as intermediate files. And then the reduce function compares each mapped face to the target, if their similarity rate exceeds the user-defined threshold, the face will be add into the ouput list.after the whole process finishes, we will get a bunch of croped face pictures and a list of original photo names.

3.1 Performance

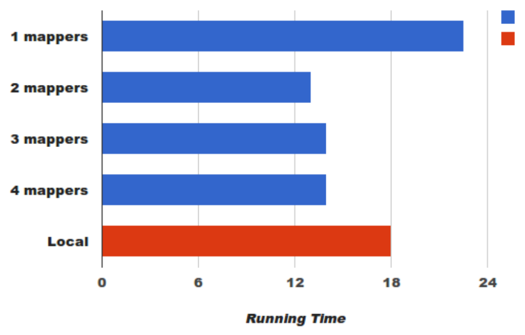

Figure 5. Overview

We have run the face recognition test for serveral times with different amount of mappers on a single server with virtual master and task servers, the result shows as figure 5. We perform a single server test as reference. When there is only one mapper, the performance is worst due to networking latency compared to the single server test. When there are two mappers, the program fully used servers' resources and hence had the best performance.

4 Conclusions and Future Work

This paper introduces how we modify the pynamo code and implement a MapReduce system based on it with python. The system can be used for all kinds of applications that other similar systems can do. We also tried a interesting way to use the MapReduce system by running a face recognition job on it.

In the future we plan to experiment with locality optimization approaches, provide more options for users such as setting minimum and maximum data size for each worker to process, developing a higher level file system based on pynamo which can split big files into small ones and retrieve all the pieces as a single file.

References

[1] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Sixth Symposium on Operating System Design and Implementation (OSDI04)*, 2004.

[2] G. D. et al. Dynamo: Amazon's highly available key-value store. In *Proc. SOSP*, 2007.

[3] leveldb: A fast and lightweight key/value database library by google. `http://code.google.com/p/leveldb/`.

[4] Opencv library – open source computer vision library. `http://opencv.org/`.

[5] Pynamo: Exploring the dynamo paper in python. `http://lurklurk.org/pynamo/pynamo.html`.

[6] Zerorpc. `http://zerorpc.dotcloud.com/`.