Cloud Programming Techniques

Barry Denby

Griffith College Dublin

April 1, 2021

Cloud Programming Techniques

- What are cloud programming techniques?
- Programming techniques that are designed to optimised cloud applications, increase efficiency, and reduce cost
- A lot of the optimisation techniques that you will have learned for general programming will still apply here

Cloud Programming Techniques

- Why is it important to program efficiently in the cloud?
  - Every instance, storage, communication, etc will cost you money
- Cost is dependant on the cloud provider you are using and also your application
- By making your applications more efficient you can handle more requests in a single instance
- Thus you may not require as many instances to serve your demand

Cloud Programming Techniques: Motivation

- Let's assume you have 10 instances that can serve 1,000 requests per minute
- A total of 10,000 requests a minute
- If you were to modify your application in such a way that instances can now serve 1,250 requests per minute then you can use 8 instances to serve 10,000 requests instead of 10
- Saves the price of renting 2 instances
- Constant battle for efficiency as demand increases

Cloud Programming Techniques: Optimisation

- What you must remember about optimisation is that you will never eliminate bottlenecks
- A bottleneck in code is always a constantly moving target
- When a bottleneck is resolved it is moved elsewhere
- The overall target is to get an application that performs good enough for the task
- The law of diminishing returns applies here (i.e. each optimisation will yield less improvement)

Cloud Programming Techniques: Development Process

- The normal software development process still applies
  - design and develop application
  - remove bugs from application
  - repeat process until satisfied
- The optimisation phase happens after and it follows a similar pattern
  - profile and look for bottleneck
  - address bottle neck and profile again
  - if improved then look for a new bottleneck, otherwise try a different approach to bottleneck

Cloud Programming Techniques

- There are multiple techniques that you can use to increase the efficiency of your requests
- And maximise the use of your resources
- There are six in total here (although there are many more)
- The first of these we will explore is the memcache

Memcache

- A memcache is a distributed cache using the main memory of multiple instances to temporarily store values
- Depending on the provider these can be dedicated instances or could be overlayed on instances running other VMs
- By using multiple nodes it is possible to increase the available memory and replicate data
- The reason we use a mem cache to temporarily store data is to reduce the number of calls to a datastore

Memcache

- The access speed of RAM is far greater than the access speed of persistent storage
- Runs on a client server architecture
- Clients must know the location of all servers
  - Usually hidden and handled by the memcache library/application itself
- Servers maintain a distributed hash table containing all the keys
  - For redundancy and reliability
  - If a memcache node goes down the others can keep the service going without losing keys

Memcache

- For an application to take advantage of a memcache it should store frequently used but rarely modified values in the cache
- The more the cache is used the more datastore requests are avoided
- If a value is not found in the memcache it must be retrieved and stored in the memcache
- Consistency is an issue as if a value changes in the datastore the cache value must be invalidated and replaced

Sharding Counters

- One of the most important aspects of the cloud is that it is parallel in nature
- Multiple requests are served at the same time
- Thus we try to avoid critical sections or contention points in our code
- One common point of contention in applications is the use of counters
  - A single variable that must be incremented by all requests
  - Must be protected by a lock
  - All requests must get this lock, update the counter, then release the lock

Sharding Counters

- As more and more requests come into the application each request must wait for access to the lock before they can finish the request
- A sharding counter takes advantage of the associative and commutative properties of addition
- By splitting a single counter into multiple copies called shards
- Which are incremented seperately and independantly by different requests
  - Each shard has its own lock
- E.g. given a counter with a value of 50 we could split it into 5 shards each with a value of 10

Sharding Counters

- Now 5 requests can update the value of the counter at the same time
- When a request wants to update the counter it is given a randomly assigned shard
  - Thus it may or may not have to wait for a lock
  - The more shards there are, the less likely a request will have to wait for a lock
- Should a request require the total value it will get all shards and will add them together
  - Of course the downside of this is that the more shards there are the more values that must be obtained and added together
  - However, this is a small price to pay for enhanced parallelism

Minimising Work

- Minimising work is a technique that is used in all applications including non-cloud application
- The idea is to reduce the overall work that an application has to do to service a request
- Smaller workload $==$ faster request
- In the case of a cloud there are a few methods that can be used to reduce the amount of work needed for a request
- We will explore two such techniques here

Minimising Work

- Replacing queries by direct key access
- By using a query you have to search for a single object and sort through all available objects
- As cloud applications tend to use key-value stores you should try and retrieve an object by key directly instead of using a query
- Saves time because you are directly accessing an object without having to search for it

Minimising Work

- Another technique you could use is to split large objects into multiple smaller objects
- An example of this is storing a file in cloud storage
- If the content of the file is stored along with the file metadata, the entire file contents must be retrieved before the request can work on that object
- The majority of time these applications are more interested in the metadata which is much smaller than the file contents

Minimising Work

- Thus if we break the object into two objects: one for metadata, one for file contents
- We need only retrieve the metadata for files first
  - Much smaller than the data contained in a file.
- After the user has selected a file for download the contents can be downloaded directly
- Reduces the amount of data needed to service a request.

Cloud for Scaling

- One of the benefits of the cloud is that an application can scale up and down in the number of instances to match demand.
- As instances can be spun up and down quickly an application should monitor itself to make efficient use of cloud resources and closely match demand
- There are two models that can be used to monitor the application.
- The approach used depends on how requests are distributed among instances in the application

Cloud for Scaling: Passive Listener

- The passive listener approach is used when requests are passed to a load balancer before being passed to an instance
- A monitor process will periodically send a simulated client request to the load balancer
- It waits to see the response time of this request

Cloud for Scaling: Passive Listener

- If the response time exceeds a threshold value it suggests that the system is under high load
  - The monitor process will then instruct the cloud system to spin up a new instance to match demand
- There is also a lower threshold value. If the response time from the simulated request is lower than this value then it suggests that there are too many instances for the demand
  - The monitor process will then instruct the cloud system to spin down an instance as there are too many being used

Cloud for Scaling: Active Listener

- The active listener approach is used when all requests are stored in a queue, and instances pull requests from that queue
- The monitor process will monitor two things
  1. The number of requests in the queue
  2. How long each request has been waiting before being served
- If the queue is getting full or the wait time is too long then the monitor process will start up a new instance to deal with the demand
- If the queue is getting empty or the wait times are becoming too short then the monitor process will reduce the number of instances

Map Reduce

- Map Reduce is a technique for working on large datasets in parallel
- Mainly designed for text based data such as logs
- Splits data into multiple chunks and distributes processing to multiple nodes.
- Models a task as a two pase compute operation based around key value pair objects
  1. A mapping phase
  2. A reduce phase

Map Reduce

- The map phase runs a filtering operation over input data
- Any lines ore text that satisfy the filter will be recorded as a key value pair.
  - For example if you were implementing the Grep tool looking for the word "foo" in the map phase you would emit a key value pair of $< foo, 1 >$ to indicate that the filter has found a part of the input data that satisfies the query
  - Keys are not required to be unique
  - All keys are written to a series of output files that are read and processed by the reduce phase

Map Reduce

- The reduce phase finds duplicate keys and collapses them into single keys
- Takes the output of the map phase as input
- How keys are combined and reduced is determined by the user of Map Reduce
  - Using the Grep example if we have 5 $< foo, 1 >$ pairs it would make sense to collapse this to a single key value pair of $< foo, 5 >$
  - Used to remove all repeat values from the result set

Map Reduce

- Because datasets in Map Reduce tend to be large
- They are distributed among multiple nodes
- The master node will try and allocate tasks to nodes that already have the necessary data stored
- This is done to reduce the amount of communication on the network and enable quicker processing of data

Synchronous and Asynchronous calls

- When serving requests there are usually many components and tasks that have to be performed
- Some of these tasks can be performed in the background i.e. enable parallel processing
- Up to now you have used synchronous calls in your use of Google App Engine
- A synchronous call is a blocking call. It will not return until the call has completely finished its work
  - Thus no other processing can be done until the call returns
  - If you have multiple synchronous called these are serviced one after another (serialisation)

Synchronous and Asynchronous calls

- Some API calls can be performed asynchronously in the background
- When an asynchronous call is made the call will start the task in the background but will return immediately
- This frees up the request to do other work including making other asynchronous calls
  - Thus parallelising work

Synchronous and Asynchronous calls

- When a request then requires the result it will make a second synchronous call to the API to get the result
- The result may already have been produced in which case it will return immediately
- Otherwise it will block until the result is obtained
- By parallelising work like this more work is done in a shorter amount of time
  - Meaning more requests can be served in a given time unit